

AD-A066 192

MOORE SCHOOL OF ELECTRICAL ENGINEERING PHILADELPHIA P--ETC F/6 9/2
USE AND EXTENSION OF AN AUTOMATIC PROGRAM GENERATOR FOR MODEL B--ETC(U)
OCT 78 J L GANA

N00014-76-C-0416

NL

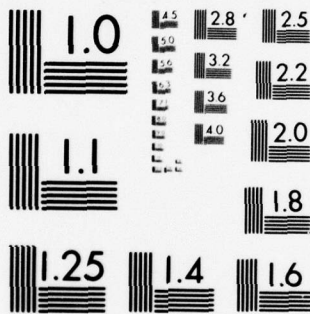
UNCLASSIFIED

78-02

1 OF 6

AD
A066192



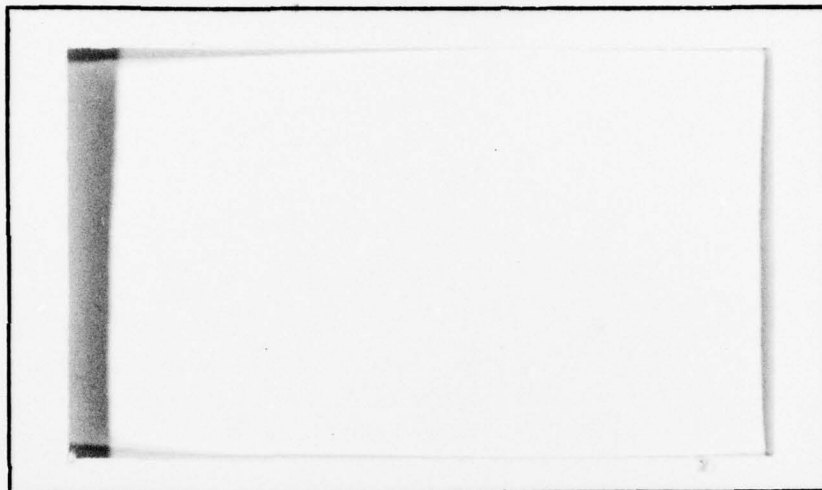


MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD A0 66192

LEVEL II

2
B.S.



THIS DOCUMENT IS BEST QUALITY PRACTICABLE.
THE COPY FURNISHED TO DDC CONTAINED A
SIGNIFICANT NUMBER OF PAGES WHICH DO NOT
REPRODUCE LEGIBLY.

DDC FILE COPY



DDC
RECEIVED
MAR 22 1979
C

This document has been approved
for public release and sale; its
distribution is unlimited.

UNIVERSITY of PENNSYLVANIA
The Moore School of Electrical Engineering
PHILADELPHIA, PENNSYLVANIA 19104

79 03 20 03 3

~~79 03 20 03 3~~

DISCLAIMER NOTICE

**THIS DOCUMENT IS BEST QUALITY
PRACTICABLE. THE COPY FURNISHED
TO DDC CONTAINED A SIGNIFICANT
NUMBER OF PAGES WHICH DO NOT
REPRODUCE LEGIBLY.**

Automatic Program Generation Project
Department of Computer and Information Science
Moore School of Electrical Engineering
University of Pennsylvania
Philadelphia, Pa. 19104

Technical Report

AN AUTOMATIC PROGRAM GENERATOR
FOR MODEL BUILDING
IN SOCIAL AND ENGINEERING SCIENCE

by

Jorge L. Gana

Prepared For
Information Systems Program
Office of Naval Research
Arlington, Virginia 22217

Under Contract N00014-76-C-0416

October 1978

Moore School Report #78-02

79 03 20 03 3



Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 14 78-02	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER 9
4. TITLE (and Subtitle) Use and Extension of an Automatic Program Generator For Model Building In Social and Engineering Sciences		5. TYPE OF REPORT & PERIOD COVERED Technical Report
7. AUTHOR(s) 10 Jorge L./Gana		6. PERFORMING ORG. REPORT NUMBER Moore School Report 78-02
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Computer and Information Science The Moore School of Electrical Engineering (D2) University of Pennsylvania Philadelphia, Pennsylvania 19104		8. CONTRACT OR GRANT NUMBER(s) 15 N00014-76-C-0416
11. CONTROLLING OFFICE NAME AND ADDRESS Information Systems Program Office of Naval Research U.S. Navy		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NR-049-153
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE 11 October 1978
		13. NUMBER OF PAGES 488 12 492p
		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Reproduction in whole or in part permitted for purposes of the United States <div style="border: 1px solid black; padding: 2px; display: inline-block;">This document has been approved for public release and sale; its distribution is unlimited.</div>		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Automatic Program Generator, Automatic Programming, MODEL III, MODEL Building Economic Studies, Non Procedural Languages, Descriptive Languages, Verification, Tolerance of Errors, Directed Graphs, Simultaneous Equations, Cycles		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) The growth in size and complexity of computer models used in econmic and engineering studies has motivated research into the use of automatic program generation techniques as a cost effective alternative. This dissertation describes this new approach to the development of software needed at different stages of model building and simulation applications. It uses the computer itself to automatically generate ad-hoc application programs, based on a description given in a high level non-procedural Module Description Language (Continued on next page) -over		

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

410 039

hpg

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

20(MODEL). A non-procedural specification uses the language of mathematics in the sense that its statements denote either assertions, which can be considered as identities or equations based in mathematical or logical relationships, or data declarations. Therefore a description does not include step by step computational instructions or control statements, nor does it contain references to the concept of memory assignments, as it is customary in programming languages. The user is required to describe the structure of his input and output data, together with the functional and logical relationships existing between them. These declarative expressions can occur in any order, and in essence they represent directly the structure of the model or system being specified, in a readable mathematical notation which does not require further translation or interpretation by the user. The automatic program generator is tolerant, in the sense that it analyzes a given specification for omissions, inconsistencies and incompleteness, attempts to automatically correct some of the misspecifications or report to the user to obtain further information until a successful and complete description is generated.

It is shown that the descriptive language is well suited for model building applications requiring the communication and integration of independently built models, and provides a mechanism for effective cooperative computation.

A Depth-First-Search algorithm was implemented in the MODEL processor to find cycles and simultaneous equations in the non-procedural specification. Blocks of simultaneous equations are then grouped together for posterior recursive ordering. Different alternatives for generation of solution procedures (in simulation and estimation) are also discussed.

A small income-expenditure model of the U.S. economy (Klein's Model I) is used to illustrate the MODEL specification needed to automatically generate an interactive simulation program. Several other examples from the field of economics are given to exemplify different features of the MODEL system.

The general objectives of this work were to use and extend the latest automatic program generation techniques in order to reduce the cost and computer oriented skills required to employ the modeling methodology. In doing so, a number of novel and useful features have been produced that contribute to a broader class of applications in which knowledge of mathematics and logic becomes sufficient, as opposed to the current demand of computer programming knowledge as well. New algorithms and techniques were also developed that enhanced the automatic generation of computer programs.

ACCESSION for

WTS	White Section	<input checked="" type="checkbox"/>
DOZ	Black Section	<input type="checkbox"/>
UNCLASSIFIED		<input type="checkbox"/>
J.S.		<input type="checkbox"/>

DATE: 11-17-71
BY: [Signature]
DISTRIBUTION: [Signature]
A 23

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

USE AND EXTENSION OF AN AUTOMATIC PROGRAM GENERATOR
FOR MODEL BUILDING IN SOCIAL AND ENGINEERING SCIENCES

Jorge L. Gana

A DISSERTATION

in

Computer and Information Science

Presented to the Graduate Faculty of the University of
Pennsylvania in Partial Fulfillment of the Requirements for
the Degree of Doctor of Philosophy.

1978

Noah S. Pinyan
Supervisor of Dissertation

A. H. Gahan
Graduate Group Chairman

Acknowledgements

I would like to express my gratitude to Dr. Noah S. Prywes for his advise and encouragement during this dissertation research, and to the Information Systems Branch of the Office of Naval Research for their generous support.

Many thanks are also extended to the members of my doctoral committee for their interest and guidance, in particular to Dr. Lawrence R. Klein for his support, motivation and the wonderful learning opportunity offered by project LINK and its members.

Appreciation is also extended to the members of the Automatic Program Generation Project for their criticism and support. To S. K. Shastri for overall help and system implementation, and to R. Reddi for programming assistance.

I would like to thank Barbara Collins for her typing of various technical reports and general secretarial assistance.

A special word of thanks to my family and friends for their encouragement, and to the memory of my father for his inspiration and sense of dedication.

Finally, I would like to express my deepest gratitude

to my wife, Maria Angelica, who provided me with infinite encouragement and joy while enduring many hardships in the course of my graduate education. Without her dedication, sacrifice, love and intelligence, this dissertation would not have been possible.

<u>TABLE OF CONTENTS</u>	<u>PAGE</u>
1. INTRODUCTION	2
1.1 Objectives.	2
1.2 General Background	5
1.3 The Approach	8
1.4 Accomplishments	13
1.5 Summary of Contributions	14
1.6 Organization of this Dissertation	18
2. BACKGROUND, MOTIVATION AND SURVEY OF RELATED RESEARCH	21
2.1 Introduction	21
2.2 Research on Automation of Software Development.	27
2.3 Summary of the Modeling Process.	32
2.4 Survey of Computer Modeling Systems	38
3. THE MODEL LANGUAGE	48
3.1 Introduction	48
3.1.1 General Background.	49
3.1.2 Overview and Intent of the Language	52
3.1.3 Main Characteristics and Capabilities	56
3.1.4 User Requirements	59
3.2 The Language Structure.	61
3.3 An Illustrative Simulation Example.	64
3.3.1 Introduction.	64
3.3.2 Description of Klein Model-I	65
3.3.3 Statement of the Problem.	68
3.3.4 Basic Concepts and Definitions.	70
3.3.5 Interfile Relationships	79
3.3.6 Relating TARGET to SOURCE Data.	81

TABLE OF CONTENTSPAGE

3.3.7 Use of Subscripts: Iteration	84
3.3.8 Use of Functions	87
3.4 Module Specification	90
3.4.1 General	90
3.4.1.1 Syntax Notation	90
3.4.1.2 Character Set	91
3.4.1.3 Operators	91
3.4.1.4 Names.	92
3.4.1.5 Use of Blanks	93
3.4.1.6 Comments.	93
3.4.1.7 Constants	93
3.4.2 Composing the Heading Definition Section . .	94
3.4.2.1 Module Name Statement	95
3.4.2.2 Source Files Statement	95
3.4.2.3 Target Files Statement	95
3.4.2.4 Reference Statement	96
3.4.2.5 Header Definition for Model I.	97
3.4.3 Composing the Data Description Section . .	98
3.4.3.1 Media Statement	98
3.4.3.2 File Statement.	103
3.4.3.3 Record Statement	104
3.4.3.4 Group Statement	105
3.4.3.5 Field Statement	106
3.4.3.6 Files Description for Model I.	109
3.4.3.7 Inter-File Relationships: POINTER type assertions.	117

TABLE OF CONTENTSPAGE

3.4.3.8 Assertions for Variable Length: LEN-type Assertions.	117
3.4.3.9 Assertions for Variable Repetitions: . .	118
3.4.4 Composing the Computation Description Section.	119
3.4.4.1 Interim Variable Description	119
3.4.4.2 Subscript Parameter Description	122
3.4.4.3 Assertions	122
3.4.4.3.1 Assertion name	123
3.4.4.3.2 Source Variable Description	123
3.4.4.3.3 Target Variable Description	124
3.4.4.3.4 Function Reference Statement.	124
3.4.4.3.5 Solution Method Description	125
3.4.4.3.6 Initial Value Description.	126
3.4.4.3.7 Test Value Description.	127
3.4.4.3.8 Assertion Statement.	127
3.4.4.4 Computation Description in Model I	131
3.5 Alternatives for Module Specification.	138
3.5.1 One-period Simulations and Residual Checks. .	140
3.5.2 Mechanisms for Dynamic Multipliers and Policy Analysis	142
3.6 Interactiveness	146
3.6.1 The Text Editor.	147
3.6.2 The HELP Statement.	147
3.6.3 Specifying Simultaneous Assertions	149
3.6.4 Reporting of Simultaneous Assertions.	151
4. THE MODEL PROCESSOR.	156

<u>TABLE OF CONTENTS</u>	<u>PAGE</u>
4.1 Introduction	156
4.2 Syntax Analysis	160
4.2.1 The EBNF/WSC Metalanguage	164
4.2.2 SAP generation by SAFG	165
4.2.3 Supporting Subroutines	168
4.3 The Storage and Retrieval Subsystem	170
4.3.1 The STORE Procedure	171
4.3.2 The RETREVE procedure.	172
4.3.3 The RETR#E Function	172
4.3.4 The RETRNAME Procedure.	172
4.3.5 The RETRPRX Procedure.	173
4.3.6 The UPDATE Procedure	173
4.4 The Precedence Matrix	174
4.5 Completeness Analysis	175
4.6 Cycles Analysis	185
4.7 Documentation.	186
4.8 Sequencing and Iteration Analysis	190
4.9 Code Generation	192
4.10 Compilation and Execution of the Generated Program	193
5. CYCLES ANALYSIS	195
5.1 Introduction	195
5.1.1 Causal Ordering.	197
5.1.1.1 Definitions.	198
5.1.1.2 Block Diagonalization and Block Triangularization	201

<u>TABLE OF CONTENTS</u>	<u>PAGE</u>
5.1.2 Graph Theory Background	211
5.1.3 Algorithms for Graph Manipulation.	215
5.2 Identification of Strongly Connected Components in MODEL.	217
5.2.1 Structure of the Precedence List	217
5.2.2 Dictionary for the Precedence Lists	222
5.2.3 Algorithm STRGCON	223
5.2.4 An Example of Application of Algorithm STRGCON	231
5.3 Linking and Merging Simultaneous Assertions.	236
5.3.1 Directory Structure	237
5.3.2 Key Names.	238
5.3.3 Storage Entries for Assertions.	239
5.3.3.1 The Common Data Area.	240
5.3.3.2 The Extended Data Area for Assertions	241
5.3.4 The LINKAS procedure	243
5.4 Tearing Strongly Connected Components.	245
6. CODE GENERATION AND SOLUTION PROCEDURES	252
6.1 Introduction	252
6.2 Solving Non-linear Simultaneous Equations	254
6.2.1 General Principles.	255
6.2.2 Selecting the Output Set.	257
6.2.3 Necessary and Sufficient Conditions for Convergence	261
6.2.4 An Improved Method.	263
6.3 Generation of a Solution Procedure.	264
6.3.1 Simultaneous Assertions in the Associative	

<u>TABLE OF CONTENTS</u>	<u>PAGE</u>
Memory.	264
6.3.2 Algorithm SOLVEGS	268
6.3.3 Code Generation.	272
6.4 Normalization.	276
6.4.1 Normalization Algorithm	278
7. MODEL LINKAGE AND COOPERATIVE COMPUTATION	287
7.1 Introduction	287
7.2 The LINK World Trade Model	289
7.3 The Long Run Model of World Trade	297
7.4 MODEL Description of MAXILONG	300
7.4.1 Specification of a Single Country Model.	305
7.4.2 Specification of the Trade Model: MINILINK.	314
7.4.3 Summary of Specification of Integrated Models.	321
8. USING MODEL FOR REGRESSION AND STATISTICAL ANALYSIS	323
8.1 General Approach.	323
8.2 Some Useful Functions	326
8.3 Examples Using Functions	334
8.3.1 An Input-output Matrix	334
8.3.2 Ordinary Least Squares Regression Module	334
8.4 Direct Implementation of Estimation Methods.	336
8.5 Further Extensions of Single Equation Methods	342
8.5.1 Principal Components	342
8.5.2 Shiller's Method and Ridge Estimators	345
9. CONCLUSIONS	349
Appendix A	356

Appendix B	389
Appendix C	404
Appendix D	424
Index	439
Bibliography	461

LIST OF TABLES

Table 5.1	Matrix for Example	203
Table 5.2	Reachability Matrix for Example	204
Table 5.3	Identification of Zero-order Elements	205
Table 5.4	Identification of First-order Elements	206
Table 5.5	Identification of Second-order Elements	206
Table 5.6	Final Ordering	207
Table 5.7	Precedence Matrix for Graph in Figure 5.4	219

LIST OF FIGURES

Figure 2.1 The Modeling Process	33
Figure 3.1 The Automatic Program Generation Process	54
Figure 3.2 Outline of MODEL Language	61
Figure 3.3 Diagram for Example: Klein's Model I	67
Figure 3.4 Hierarchical Model for Time-series Record	71
Figure 3.5 Hierarchical Data Description Statements in MODEL	74
Figure 3.6 Data Network of Source Files in Disk	75
Figure 3.7a Assertions for Title Heading	82
Figure 3.7b Assertions to Compute Year and Period of Simulation in Actual Period NP for File REPORT	82
Figure 3.7c Income Identity in Terms of INTERIM Variables. Assertion is Subscripted with time Variable T	82
Figure 3.8a Equivalent References for a Repeating Field	85
Figure 3.8b Consumption Function in Model I with Subscripts	85
Figure 3.8c Use of Subscripts in Extended Form of an Assertion	85
Figure 3.9 Header Definition for Model I	96
Figure 3.10 Description of PARAM File for Model I	108
Figure 3.11a Description of File Bank	110
Figure 3.11b Sample Entry for Time-series Corresponding to First Variable: Consumption	111
Figure 3.12 Description of Documentation File DOCUMENT and Target Solution File CONTROL	112
Figure 3.13a Description of Files in Terminal	114
Figure 3.13b Assertions for Files in Terminal	115
Figure 3.14 Use of Interim Variables in Model I	120

Figure 3.15	Diagram of BANK File Depicting Computation Of OFFSET for Solution	131
Figure 3.16	Auxiliary Assertions for Generality of Module Specification	132
Figure 3.17	Relating Interim Variables to Predetermined Data	133
Figure 3.18	Model I Characteristic Equations	134
Figure 3.19	Relating Target Data to Interim and Source Data	135
Figure 3.20	Control Solution Documented with new Record	136
Figure 3.21	Module Specification Using Array Notation	138
Figure 3.22	Inner Iteration Description	149
Figure 3.23	Sample Formatted Report from Model I Example	153
Figure 3.24	Automatic Grouping of Simultaneous Assertions in Model I	154
Figure 4.1	Overview of the MODEL Processor	157
Figure 4.2	Block Diagram of the Syntax Analysis Phase	159
Figure 4.3	First Pass of SAPG	165
Figure 4.4	Flow Diagram of Completeness Analysis Phase	177
Figure 4.5	Growing [P] for Model of Income Determination	179
Figure 4.6	Directed Graph for Model of Income Determination	180
Figure 4.7	Statements for Model of Income Determination	181
Figure 4.8	Example of Cross-Reference and Attribute Report	187
Figure 4.9	Example of Precedence List Report	188
Figure 4.10	Summary of the Sequencing and Iteration Analysis Process	190
Figure 5.1	Digraph of the Matrix in Table-5.6	207
Figure 5.2	Graph G with Fragment F	211
Figure 5.3	Condensation of Graph G in Figure 5.2	212

Figure 5.4 Directed Graph Representation for Two Assertions in MODEL	218
Figure 5.5 Flowchart for Algorithm STRGCON	225
Figure 5.6 Digraph with Strongly Connected Components	230
Figure 5.7 Precedence Matrix Generated for Sample Equations	231
Figure 5.8 MODEL Graphic Representation of Sample Equations	232
Figure 5.9 Jungle Generated by DFS Algorithm	233
Figure 5.10 Identification of Strongly Connected Components by MODEL	234
Figure 5.11 Undecomposable Adjacency Matrix	246
Figure 5.12 Transition Graph	247
Figure 5.13 Semidecomposition Illustration	248
Figure 6.1 Convergent Solution	258
Figure 6.2 Divergent Path	259
Figure 6.3 Simultaneous Assertions in the AM	265
Figure 6.4 Extended Data Area for Assertion	266
Figure 6.5a Code Generation: One Storage Entry for Each Assertion	269
Figure 6.5b Code Generation: Subgroup Under Single Storage Entry	269
Figure 6.5c Code Generation: Same Name and Common Qualifier	269
Figure 6.6 Code Generation Schema	273
Figure 6.7 Symbolic DT for an Assertion	278
Figure 6.8 Normalized DT Corresponding to Assertion in Figure 6.7	280
Figure 6.9 PL/1 Code for Normalization Algorithm	284
Figure 6.10 Example of Normalization for a DT	285
Figure 7.1 Schematic Diagram for LINK System	295

Figure 7.2 Block Diagram for Country "A"	301
Figure 7.3 Block Diagram of Linked Trade Model MINILINK	303
Figure 7.4 Specification of Country "A" Reduced Economic Model	306-310
Figure 7.5 Specification of MINILINK	316-319
Figure 9.1 Canonical Block Angular Form	351
Figure 9.2 Proposed MODEL System Development	355

LIST OF ALGORITHMS

ANALCMP

PL/1 Code	403
Block Triangularization	202
Code Generation	
SOR	273-274
LINK	289-292
LINKAS	242,398
Normalization	
Description	277-283
PL/1 Code	284

PRECED

PL/1 Code	401
SOLVEGS	267-270

STRGCON

Description	222-224
Flowchart	225
PL/1 Code	226-228,392

CHAPTER 1

Introduction

1.1 Objectives

This dissertation describes a new approach to the development of software needed at different stages of model building and simulation applications. It uses the computer itself to automatically generate ad-hoc application programs, based on a description given in a high level non-procedural language. A non-procedural specification uses the language of mathematics in the sense that its statements denote either assertions, which can be considered as identities or equations based in mathematical or logical relationships, or data declarations. Therefore a description does not include step by step computational instructions or control statements, nor does it contain references to the concept of memory assignments, as it is customary in programming languages. The user is required to describe the structure of his input and output data, together with the functional and logical relationships existing between them. These declarative expressions can occur in any order, and in essence they represent directly the structure of the model or system being specified, in a readable mathematical notation which does not require further translation or interpretation by the user. The automatic program generator is tolerant, in the sense that it analyzes a given specification for omissions,

inconsistencies and incompleteness, attempts to automatically correct some of the misspecifications or report to the user to obtain further information until a successful and complete description is generated.

The most important advantages in using the system reported in this dissertation is the ease in integration and expansion of specified models.

The general objectives of this work were then to use and extend the latest automatic program generation techniques in order to reduce the cost and computer oriented skills required to employ the modeling methodology. In doing so, a number of novel and useful features have been produced that contribute to a broader class of applications in which knowledge of mathematics and logic becomes sufficient, as opposed to the current demand of computer programming knowledge as well. New algorithms and techniques were also developed that enhanced the automatic generation of computer programs.

The goals of this research can be summarized as follows:

- 1) To develop a general non-procedural language and its processor which could act as the foundation for a broad class of applications requiring the generation of ad-hoc computer programs for use at different stages of model development.

- 2) To design the system with the concepts of

"sharing" and "integrability" of both data and computational description of processes, which could permit the communication and integration of individual models at a high descriptive level.

3) Efficiency of the system and openness to innovations.

These very general overall objectives encompass a variety of new concepts not found in other computer modeling facilities. First it implies the use of a declarative language with standard mathematical notation, for non-procedural specifications of models. Programming languages in contrast contain nonmathematical features such as transfer and assignment statements. The language of mathematics is familiar to model builders since it is used to describe the models, therefore it is convenient to employ an isomorphic description for computer processing. Second, the processor of the system is similar to a high level compiler whose object code is a computer program written in a suitable high level programming language (PL/I). Therefore the computer itself is used directly as a "program writer", in a sense replacing the application programmer interface with the consequent reduction of errors due to the communication and programming of the process. Also, since the required programs are generated in a high level programming language, these can be processed by any other facility possessing a compiler for such object code.

Therefore the generated modules are transportable.

1.2 General Background

With the widespread use of computer models in economic studies, engineering, biological and social sciences, it is increasingly important to develop appropriate software tools and techniques to facilitate the application of this methodology. The formulation, simulation, refinement and experimentation phases in the development of complex systems should be performed without undue restrictions, in an adequate time frame and at a reasonable cost to be effective. However, in spite of recent advances in computer hardware technology, which makes possible the processing of larger and more complex models quickly and accurately, the main factors affecting the use of computers by model builders still remain as follows:

- 1 - The cost of describing the model and associated solution procedures in a computer language. This task is usually handled by an application programmer, unless the host language is a special purpose modeling language oriented to the user particular application.
- 2 - The cost of formatting and introducing the input data required for the problem solution into the system. In some cases this task can be performed by clerical staff who can keypunch or type the data directly into the system. However in many instances input data originate

from different sources and may be received in different formats, sometimes requiring a more complicate translation process, and again the need for expertise in computer programming.

- 3 - The cost of maintaining the system. Model building is a dynamic activity which requires constant revisions and updates of both the structure of the models and their data. Normally the host software system will provide the necessary tools which allow the user himself (the model builder) to maintain his model. However it is usually necessary to be familiar with computer processes for this activity, and often the host system itself require modifications for handling exceptions such as new solution or estimation techniques, or experiments not in the original discourse of the host language. Depending upon the complexity of the host modeling system this process may require an application programmer or a team of system programmers.
- 4 - The cost of actually running or executing the required process (transformation, estimation, simulation) by the computer. This stage requires practically no human intervention and will depend on both hardware configuration at hand and efficiency of the host system and solution procedures.
- 5 - Finally , consideration must also be given to the costs involved when testing and debugging a model. Considerable time and expertise are devoted to this

stage analyzing the program logic, the model specifications, etc.. The larger and more complex the models, the more difficult it is to trace faults, both in the program logic and in the model specifications and interactions.

Various computer modeling systems have been developed recently that facilitate the interface between model builders and computers at different stages of the process. In general, the "best" system for a particular application will depend on user needs and considerations about the cost factors mentioned before, as well as availability, ease of learning, etc.. However, as the survey in Chapter 2 will show, despite the overall improvements these facilities provide, they are far from solving a number of problems imposed by the use of larger and increasingly complex models.

As systems increase in size, data management becomes a major undertaking requiring the manipulation of more complicated data structures. Modeling languages on the other hand, impose a rigid structure and organization to the user, leaving only high level programming languages possessing data description capabilities, such as PL/1 or COBOL, to handle those cases. This situation forces the user to interface with middle level application programmers to develop and maintain the host system and his model.

Another major problem is the lack of generality found in most modeling systems to handle new situations without a major programming overhaul. Recent developments show the need to integrate independently builded models. For instance in economics, the sytems that are being modeled do not stand alone, and regional or world wide integration of existing models is becoming necessary. Due to the complexity of the methodology, the exchange and interface of models from different originations amongst developers has led to misunderstandings and difficulties in the process.

1.3 The Approach

Towards the proposed objectives, this dissertation has contributed in the design and implementation of a non-procedural Module Description Language, MODEL, and its processor. The system development was carried out in cooperation with the Automatic Program Generation group at the Moore School. The research reported herein expanded the language to satisfy the proposed goals and implemented the necessary additions to the MODEL processor which permits the specifications of models and their associated solution methods. The Automatic Program Generation group was responsible for implementing the core of the system, particularly in relation to extended capabilities to generate additional program structures and methods of optimization, and of improving the man/machine interface to decrease the manual labor and skills required to enter a

given specification [SHA 78, MOT 77].

The MODEL language permits the user to specify his model and associated data and solution methods by a set of declarative or descriptive statements. The specification consists of the following types of statements:

- Module Description
- Data Description
- Assertions

The statements in the "Module Description" section identifies a particular model and the associated input or "source" files, as well as the desired output or "target" files. This section may also include references to other data or computational descriptions previously stored in the MODEL "data base", thus allowing the sharing of complete or partial specifications. The "Data Description" statements declare the structure, type and organization of data items in each file. Finally the "Assertions" describe various types of data interrelationships.

The statements consist of independent descriptions that can be submitted in any order. This allows the user to build a given specification modularly, with the declarations added as they become available. There are no control structures in the language, since the sequencing and control logic of the object program is deduced automatically by the processor. Nor it is possible to specify computer concepts

such as counting, input-output commands, control code, memory assignments or other typical computer programming concepts.

Since MODEL deals only with information concepts and relationships, it is application independent, containing only computer programming "knowledge". Therefore, it should be possible to extend its domain to a broader spectrum of applications.

The use and components of the MODEL language are explained in detail in Chapter 3.

The MODEL processor takes the specification of a module in the MODEL language, and performs the following tasks:

(1) Syntax Analysis - In this first phase the given specification is analyzed, one statement at a time, to detect syntactic errors and some semantic problems. After successful analysis the statements are stored in a simulated associative memory for ease in later retrieval and analysis.

(2) Determination of Precedence Relationships - The unordered statements in the associative memory are analyzed to determine precedence relationships between each data name used in the specification. The relationships are used to construct a precedence graph to be used in subsequent analysis for

completeness and correctness of the specification and for sequencing before the generation of the object code.

(3) Analysis of the Directed Graph - In this phase the graph is analyzed for inconsistencies, incompleteness or cycles. During this phase the processor endeavors to modify redundant or ambiguous statements, or to compose missing descriptions using the information at hand and default parameters. If necessary it solicits from the specifier additional information or modification of his specification. The cycles analysis stage is one of the contributions of this dissertation that permits the efficient solution of large scale simultaneous equation models. Essentially the graph representation of a given specification should not contain circular definitions or cycles for proper sequencing, unless they are part of a simultaneous equation system which can be interpreted as recursive blocks requiring iterative solution techniques. The cycles processor therefore analyzes the graph for cycles and proceeds either to combine nodes corresponding to simultaneous equations for later solution, or gives the user an error message and ask him to open the existing loops in the specification. Large scale models can use the cycles processor to automatically partition (decompose) the

specification into its block recursive components for both efficient solution, and for a better understanding of the interrelationships between elements of the model. Chapter 5 explain in detail this component of the processor.

(4) Documentation - This phase generates some reports from the analysis of the MODEL specification; a cross-reference table of all names provided by the user and a complete (possibly augmented or modified) formatted MODEL specification of the problem. This clean copy is also saved in a MODEL statement library for later reference.

(5) Sequencing - The MODEL statements are ordered using standard topological sorting algorithms as applied to the directed acyclic graph representation of the problem specification. A flow-chart like report is generated representing the control logic of the module.

(6) Code Generation - Using the flow-chart entries, the program for the module is generated in the high level programming language PL/I. Necessary input-output commands are inserted together with other control and procedure call statements. Different solution methods or simulation and estimation techniques will be automatically generated from the MODEL description, as described

in Chapter 6 of this work.

(7) Compilation and Execution of the Generated Programs - Finally the program is submitted for compilation by the PL/I optimizing compiler, which produces an object or load module for integration of the program into the system and subsequent execution by traditional means.

A more detailed overview of the MODEL processor and its components is given in Chapter 4.

1.4 Accomplishments

The system described in this dissertation in conjunction with several examples of its application to model building in economics demonstrate the feasibility of automating software development in every stage of model building, therefore effecting a direct communication between the model specifier and the machine. This effort was an outgrowth of interdisciplinary and cooperative research in which this author draw experience by implementing the host information system for solving the LINK world trade model [GAN 76], and participated in the design of a number of other computer modeling facilities, while interfacing with the development of MODEL in the Automatic Program Generation group. The outcome of this effort is a processor with immediate potential for saving man-hours in software development and for increasing the efficacy of the modeling process.

Furthermore, as Chapter 7 will illustrate, the automatic program generation methodology, in conjunction with other technological innovations in computer sciences such as distributed computation and networking, plus data base management techniques, could be used as the nucleus from which applications requiring cooperative computation could evolve by allowing the sharing of both human and computer resources at a higher level of abstraction.

1.5 Summary of Contributions

The adaptation of an automatic program generation system to model building has contributed essential and major improvements to both the methodology followed in the application area, as well as to the techniques and algorithms used for automatic generation of software from a general non-procedural language. The use of a formal descriptive language by the model specifier impose a certain discipline, but allows the representation of the problem for processing in a natural and logical manner, relieving him from tedious and expensive time and efforts dedicated to program debugging. The needs for innovative and complex techniques of completeness and consistency analysis of subscripted expressions found in the specifications of many problems areas was in part unveiled while trying to generalize the MODEL language domain to model building, in particular to dynamic situations involving lagged variables. The solution to the general problem is reported in the

dissertation by Shastry [SHA 78] who also implemented the core of the MODEL processor.

The specific achievements of this dissertation research, beyond the previously reported activities are summarized below:

- As a generator of modeling software facilities, MODEL provides the user with a number of new features not found in other state of the art modeling systems. These include:

(1) Generality: the user can generate ad-hoc program modules according to his needs. It is also open to innovations.

(2) Non-procedural Language Interface: allowing direct communication of the specification without control or computer programming concepts.

(3) Generalized Data Description Facilities: provides a method for data and program independence.

(4) Sharing and Integration of Modules: providing an effective environment for cooperative computation.

(5) Transferability: models and their solution procedures are generated in the high level PL/1 code, therefore they can be transported and processed in other centers under a PL/1

compiler.

(6) Efficiency: several levels of optimization are in effect; in code generation, memory allocation, sequencing as well as in solution procedures. Also the fact that descriptions are not interpreted, but instead code is generated for later execution, imply savings in execution time for many applications and repetitive experiments.

- The implementation of a new cycles algorithm in the MODEL processor provides an effective and efficient way to deal with large and complex models containing simultaneous interactions or feedbacks. The algorithm utilizes well known linear graph search procedures, however its implementation as a partitioning technique in a totally non-procedural specification, with additional man/machine effective interface is novel. A number of new extensions to the basic algorithm, which provide for additional decomposition of a complex structure are also discussed in Chapter 5 and were experimented on as part of this work.

- The methodology for automatic generation of solution procedures for simultaneous equation systems, in conjunction with a new normalization algorithm will allow the user to concentrate on his specification instead of on complex programming of solution methods. The

generated solution algorithms are therefore transparent to the user and he is only requested to provide the associated solution parameters when other than the processor supplied default values.

Also a generalized approach to regression and statistical analysis is discussed in Chapter 8, and again the user is only required to provide the mathematical description of his estimation technique, with which he is familiar. Code will be automatically generated based on the functional description of requirements. A comprehensive set of matrix and vector operation functions are provided to that effect.

Although a complete test of the MODEL processor was precluded, due to the fact that some of its phases including sequencing, code generation and memory optimization are in the developmental stage, Appendixes A and C contain listings of simulation examples processed by MODEL previous stages. The feasibility for the complete automatization of the model builder software needs are nevertheless evident.

In an effort to generalize the basic MODEL language, no attempt was made to orient the language to a particular application. As the conclusion chapter will show, it is also feasible to build a more natural interface between the user and MODEL, which could allow a better communication by using a terminology familiar to specific application areas, as well as a question-answering system for overall guidance

in composing the specification. Such a facility, in conjunction with the "programming knowledge" and tolerance embodied in the present MODEL system will greatly benefit model builders by allowing effective utilization of manpower and machines.

1.6 Organization of this Dissertation

Chapter 2 gives the background and motivation for this work together with a survey of related literature and research. It reviews both efforts and research in the automation of software development, as well as those aimed at simplifying the task of model building and simulation, placing them in perspective.

Chapter 3 gives a detailed definition of the MODEL language, its structure and syntax. It can be used as a reference manual for a prospective user. After an overview of the language capabilities, some basic definitions and concepts in information organization are given. The use of the language is illustrated through the specification of a simple but comprehensive simulation example from the field of economics.

Chapter 4 provides an overview of the MODEL processor and its components. It summarizes the methodology and algorithms used in each phase by the program generation system.

Chapter 5 describes the algorithm and programming techniques used in the Cycles Analysis phase. The needs for model decomposition into recursive blocks are explained with examples and the methodology for further "tearing" or semi-decomposition of recursive blocks is explored.

Chapter 6 presents the background and necessary programming schemes for the automatic generation of iterative solution procedures. It also provides a normalization mechanism for future implementation in the MODEL processor.

In Chapter 7 the concept of cooperative computation is introduced and the "sharing" and "integration" capabilities of the MODEL system are illustrated by describing a condensed form of the LINK world trade system which is used as an approximation of the more complex simulation model.

Chapter 8 demonstrates how to use MODEL in regression and statistical analysis, giving several examples of specifications using assorted functions in the system. A more general approach is also discussed and proposed for future implementation.

Chapter 9 summarizes the overall conclusions that can be drawn from this research, and suggest directions for future investigation and development.

The output listings produced by the MODEL processor from the sample simulation example introduced in Chapter 3 is

given in Appendix A. Also a sample multicountry integrated model corresponding to a reduced approximation of the LINK world trade model presented in Chapter 7, is given in Appendix C. The reports correspond to documentation produced up to, but not including the sequencing and code generation stages.

Two other appendices are included with the programs composing the Cycles Analysis processor and a list of functions included in the MODEL library for general regression and statistical analysis.

CHAPTER 2

Background, Motivation and Survey of Related Research

2.1 Introduction

The motivation for the research reported in this dissertation arises from concern over the growth in size and complexity of computer models used in simulation studies. As models become larger, the difficulties in their estimation, programming and solution tend also to multiply, and so does the requirement for trained application programmers to keep up with data management and maintenance demands. In addition to these costs related to software development and information systems support, it is necessary to mention the major problem that complexity brings about: it becomes increasingly difficult for the user (the model builder) to understand the structure of the model and its interactions as it grows, and to manipulate it as well.

The use of a non-procedural language to automatically generate application and solution programs, as well as necessary software systems support for all or selected stages of model building is expected to solve or ameliorate current difficulties through the following special features:

- 1.- Generality; it is possible to tailor the generated programs for specific needs and data structures.
- 2.- Simplicity; it does not require traditional programming knowledge from the user, therefore users

from many disciplines can be attracted to generate complex programs without the need of extensive training in programming languages and without having to communicate requirements to an application programmer.

3.- Abstraction; since a non-procedural language is closely related to the language of mathematics, the specification of a model and its data processing requirements becomes adequate for both human interpretation and communication, as well as for computer processing of the same statements. A description in a non-procedural language does not contain control statements nor is sequencing of statements necessary, therefore are better suited to describe complex systems than procedural languages which do not possess the same level of abstraction.

More detailed characteristics particular to the MODEL language are given in chapter 3.

Although the primary orientation of this work has been toward solving programming problems currently faced by model builders in the field of economics, various other disciplines in the social, engineering and biological sciences share similar characteristics. Soylemez's [SOY 71] development of a process unit program generator represents an attempt to further automate computer aided chemical process design calculation procedures, as part of a more

ambitious goal for the automation of the entire design for an arbitrary chemical plant. The emphasis of the generator of process unit programs is on symbolic equation analysis and ordering techniques. The language is a "problem oriented" programming language designed to accomodate chemical engineers. The program generator was implemented in FORTRAN and closely relates to programs designed to manipulate symbolic information (i.e., algebraic substitution and rearrangement). It gives extensive bibliography of related chemical engineering work.

Other applications may require implementation of additional techniques and/or solution procedures from those reported in this dissertation, for instance Garfinkel et al. [GAR 77] reports on the necessity of using an heuristic search technique in dealing with the construction of complex metabolic models, since it is necessary to iteratively revise the model until it approaches acceptability. Because of the large number of unknown variables, and insufficient data to uniquely determine or best estimate them, the search space in the iterative phase becomes so large that methods from the field of artificial intelligence become a necessity.

In economics, with the widespread applications of econometric models in forecasting and policy analysis, the techniques of automatic program generation seems the most appropriate in order to reduce rising costs of model

development and facilitate experimentation with them. Although the size of an econometric model for a modern economy will depend on the amount of disaggregation sought, a minimum of 25 to 50 equations is considered standard. The most successful models of the U.S. economy experimented a growth in size from an average of 20 to 50 equations in 1950-1955 to 200 equations by 1965-1970 as exemplified by the Wharton Model at the University of Pennsylvania, the MPS model (MIT-PENN-SSRC) and the Brookings model. Currently some of these models are using 500-1000 equations and the number of variables is expected to double with the tendency to integrate national macro with sectoral models.

Many models are still maintained at the universities where they originated. Econometric groups at the University of Pennsylvania, University of Michigan, Princeton University and UCLA produce regular forecasts. Also government agencies have developed and used several models of the U.S. economy; for instance, the model of the Bureau of Economic Analysis of the Department of Commerce is one of the more sophisticated. The use of econometric models by private corporations has also proliferated in recent years.

The international scenario is equally active in the model building activity, with most industrial countries, in particular Japan, Canada and the United Kingdom, following the lead of the United States. Developing countries, despite deficiencies in their statistics, are also "taking

off" and either building or using econometric models for development planning. Some of these models originate in the countries itself, others in different universities or international organizations. For example the United Nations Commission for Trade and Development (UNCTAD) prepares a series of regional models that cover the so called "third world". Also the Centre for Development Planning, Projections and Policies (CDPPP) of the United Nations has developed a series of models for the centrally planned economies of eastern european countries and a model of the Soviet Union has been jointly developed by Stanford Research Institute and Wharton Econometric Forecasting Associates at the University of Pennsylvania.

The need to integrate models is also a relatively recent development. The systems that are being modeled do not stand alone, and regional or world-wide integration of independently developed models is becoming necessary.

The development of current integrated multinational models had their roots in the early works of the american economist Metzler [MET 50], who first introduced a mathematical formulation for a multi-regional model. Since then a number of models of the world economy have been built (see, for instance, a survey by Taplin [TAP 67], and more recently the bibliography given by Berner [BER 77] in his study of merchandise trade among the countries of the European Economic Community (EEC)). Of these, the most

ambitious and demanding in terms of information processing resources is project LINK [BAL 73]. This is an international cooperative effort to model world trade and study the international transmission mechanisms. It is organized as a conglomerate of econometric research groups from all over the world, which are responsible for development of individual country or regional models to be integrated. The whole aggregate of models is simultaneously solved at the LINK center at the University of Pennsylvania for consistent forecasts of world trade. The process implies formidable calculations. Standardization of individual country models is taking place in a number of dimensions. However, the philosophy of the project is that individual member countries are responsible for their models and are in the best position to judge particular structures and specifications. There is no attempt being made to follow a particular model pattern for each individual country. This implies huge amounts of resources dedicated to handle the data management, data manipulation and computational problems involved. Chapter 7 describes the LINK system in more detail, as an example of cooperative computation, and illustrates the benefits of the Automatic Program Generation approach to model building.

Despite the orientation of this dissertation to the solution of econometric models, the MODEL language, which accepts non-procedural specification of models and their data and solution requirements, is not restricted to any

particular application area, as long as the relationships between variables depicting its structure are described by algebraic equations. Since there are no assumptions regarding data structures (eq. time-series or cross-sectional data) nor specific application oriented keywords in the language, MODEL can be effectively used as the basic foundation for a broad class of applications where the automatic generation of programs becomes cost effective. The generality can be achieved by proper introduction of functions and solution procedures as exemplified in chapters 6 and 8 of this dissertation.

2.2 Research on Automation of Software Development

This section summarizes different approaches to automate the software development process and places them in perspective. Traditionally the system development process has been divided into the following phases [TEI 71, COU 73, PRY 74]:

- (1) - Determination of problem requirements and demands for information.
- (2) - Functional specification of the system.
- (3) - System design: modularization, interfaces, parametrization.
- (4) - Program design.
- (5) - Program implementation: coding, debugging and testing.

- (6) - System integration and installation.
- (7) - Operations and maintenance.

The idea of systematically using the computer to automate some of these phases, and therefore reducing labor costs and human errors, developed early with computer usage. This is exemplified by the appearance of such facilities as assemblers, compilers, routines and procedures, macros, operating systems etc.. Historically these developments have proceeded in a "bottom-up" fashion as stated by Prywes [PRY 77a]:

"First to be automated were the layers associated with execution of operations. This was followed by automation of the layers associated with language translation and program optimization. Recent emphasis has been placed on improving efficiency and utility of programming methodology and high level programming languages. The chronological order has been dictated by the use of the bottom layers in automation of each new layer. Also, subsequent improvements in any one layer were quickly utilized in the layers above it..."

The rate of progress in the automation of the first two phases has been much slower in relation to that experienced by the lower ones. This is due to the nature of the problems, which require a knowledge of the application area and problem solving techniques. The state of the art in artificial intelligence methodology has not yet developed to the point where a computer could generate a knowledge data base from an interaction with the user in natural language, unless the problem domain is a very restricted one [HEW 71,

WIN 72, SHA 73, PRY 74]. A more promising approach seems to be the incorporation of "expert systems" for particular areas into the computer, which can orient the user, through an interactive session, to specify his problem, resolve inconsistencies and propose alternative solutions. An example of such a system developed at the University of Pennsylvania for automatic testing of electronic equipment will be described later in this section.

Manual aids to facilitate systems analysis have been available in forms-oriented languages [ADS 68, TAG 68] and later processors were built for them [THA 71, IBM 71]. These efforts were mainly oriented to facilitate phases (2) and part of (3).

Several contributions have been reported on the automation of the physical design of the system (step (3) above). For instance models exist for the selection of file organization for a simple system [SEV 72] and for the simulation, evaluation and selection from a range of alternative file organizations [CAR 73]. A general automatic file and module design system has been proposed recently by Gibb [GIB 75].

Two projects have tried to encompass the whole of the systems development process: the ISDOS project at the University of Michigan lead by Teichroew [TEI 71, TEI 72, HER 73, TEI 74], and the Automatic Programming Group at MIT. The ISDOS project developed a system which analyzes

functional systems specifications in a Problem Statement Language (PSL). A Problem Statement Analyzer (PSA) subsequently checks completeness and consistency and produce numerous documentation reports. Nunamaker expanded the PSL/PSA system to include automation of physical system design and program coding [NUN 69, NUN 71, NUN 72a, NUN 72b]. An application involving the development of a financial management system using the integrated tools of ADS, PSL/PSA and Nunamaker's SODA (System Optimization and Design Algorithm) system was reported in [NUN 76].

The efforts at MIT are integrated in PROTOSYSTEM I which consists of a top "expert system" which performs the duties of an automated consultant to the management of a distribution organization [HAX 75, MAR 74, MAL 75, BOS 76]. The bottom part of PROTOSYSTEM I receives the specifications from the top part and performs systems design and generates PL/I code [RUT 76].

The Automatic Program Generation Project at the University of Pennsylvania has conducted research since 1973, when a data description/manipulation language was developed to automate the generation of data conversion programs [RAM 73]. Unlike the other projects mentioned above, the emphasis of this group has been on a "bottom-up" research priority in agreement with the chronological developments and economic demands of software as described before. These efforts have culminated in the development of

the NOPAL and MODEL APG systems.

The NOPAL system was developed for designing functional and fault isolation tests of analog electronic equipment and for generation of corresponding programs for computer controlled automatic test equipment [PRY 75a]. The system consists of two independent parts: the top part developed by Tinaztepe [TIN 77] determines efficient test specifications expressed in the NOPAL language. The bottom part developed by Chang [CHA 77] accepts these specifications and produces as output an efficient test program in a procedural test programming language OPAL. The bottom part can also accept test specifications directly from the user.

The MODEL system is a bottom-part only processor, oriented primarily to phases (4) and (5) of the system development process. The objective has been to obtain greater generality of APG systems progressively through development of a bottom part for a broad class of applications and requiring reduced user proficiency levels. The first version generates PL/1 programs for business transaction processing applications [RIN 76]. A revised version, called MODEL II, is an operational system with simplified user language [PRY 77b] and have recently been used for an application involving the production of programs needed to extract and transform a variety of data from the Internal Revenue Service (IRS) tape program [PRY 77c].

The research reported in this dissertation has extended

the domain of applicability of MODEL to include facilities for generating programs needed in modeling simulation and forecasting, and in general has oriented the system to engineering and scientific professionals. The effort was carried out in conjunction with the development of a new version called MODEL III, which includes interactiveness, more sophisticated completeness and iteration analysis [SHA 78], and in general capabilities to generate additional program structures and methods of optimization [MOT 77].

The potential benefits of automating each of the phases of the system development process was evaluated in a survey by Prywes [PRY 74]. Although different applications can be affected quite differently by the techniques of APG systems, continuation of present trends in software-hardware costs, which were set at 2:1 in 1970 and conservatively estimated to be 10:1 by 1985, as well as raising demands for skilled labor force needed in development of complex systems, indicates the use of computers in the automation of systems development as a viable alternative.

2.3 Summary of the Modeling Process

Figure 2.1 shows schematically the steps involved in the process of model building. The divisions are similar to those described by Naylor et al [NAY 68]. It starts with the collection and processing of real world data from many sources. These may include field surveys, questionnaires, samples, documents from governmental agencies or

international organizations, physical experiments, engineering records, etc.. This task is normally tedious and time consuming. It involves identification, auditing, editing, assignment of codes, recording and verification.

Often the first recording medium is not suitable for data processing applications, so it is further translated and converted into a data bank where manipulations are performed.

The Data Analysis phase can be divided into:

(a) - Preliminary data analysis - This stage calls for the performance of such operations as sorting, collating, merging, information retrieval, transformations and generation of new variables, as well as the representation of subsets in tabular forms or graphic plots. It is intended here to familiarize the user with the data and to allow him to manipulate and organize it in a manner which is convenient for the model structuring.

(b) - Simple statistical data analysis - The user can obtain here initial indications of dependencies between variables from calculation of simple or multiple correlations among data variables. This phase has the function of confirming the user's a priori theories about certain structures or transformations of the data to be applied in subsequent modeling stages. Other typical operations can include computations of means, standard deviations and covariances.

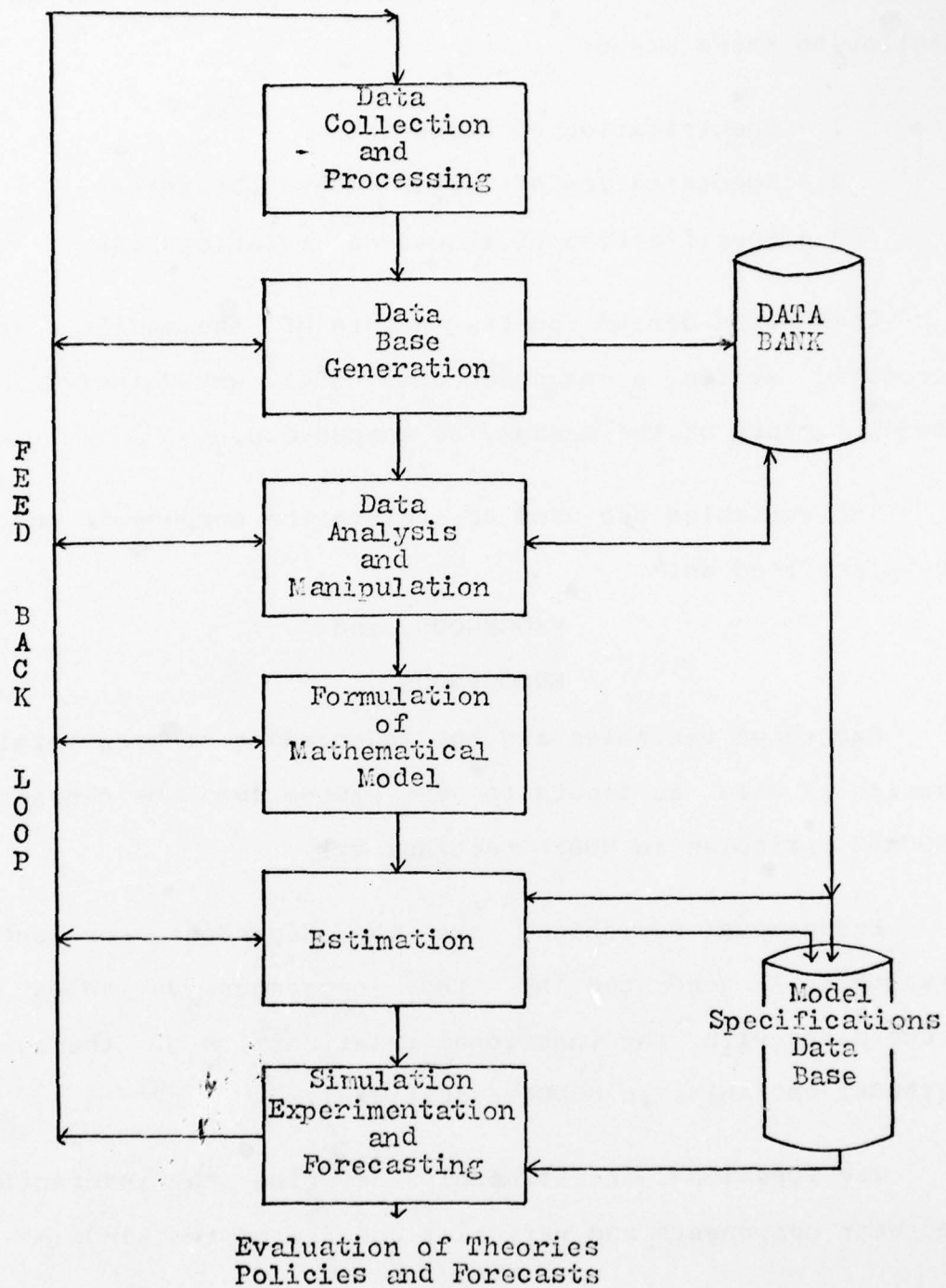


Figure 2.1
The Modeling Process

The formulation of a mathematical model consists of the following three steps:

- 1 - Specification of components
- 2 - Specification of variables and parameters
- 3 - Specification of functional relationships

Components depend on the nature of the model. In an economic system, a macroeconomic model would include the major sectors of the economy as components.

The variables are used to relate the components and can be classified as:*

EXOGENOUS, and
ENDOGENOUS

Exogenous variables are the independent or predetermined variables used as inputs to the system (can be considered SOURCE variables in MODEL terminology).

Endogenous variables are the dependent or output variables, generated by the exogenous variables in accordance with the functional relationships of the system (TARGET variables in MODEL assertions).

The functional relationships describe the interactions between components and variables and are of two kinds:

* A more refined classification including state variables and other elements is also possible.

IDENTITIES, or

STOCHASTIC (Behavioral, technical or statutory)

Identities are either definitions or tautological statements about the components of the model.

Stochastic equations are hypotheses or mathematical equations relating the model's endogenous to its exogenous variables. If the equations describe the behavior of various groups in the economy, like in the case of an investment function which shows the relationship between investment expenditures to various explanatory variables, it is called a behavioral equation. Other equations describe technological or institutional relationships, for instance a tax function showing the relationship between tax revenues to various factors affecting it. All these relations are empirically testable.

Whether a particular variable is classified as exogenous or endogenous depends on the particular objectives of the model and other factors. It is possible to change the status of a variable first classified as exogenous to endogenous, after a suitable explanatory equation is incorporated into the system.

The specification of the elements is by no means an easy task. The process of observing some system in the real world, formulating some explanatory hypothesis and abstract from them the formulation of a mathematical model are still far from being a regular procedure with easy rules.

Once the user has formulated a number of mathematical models describing the behavior of the system, it is necessary to estimate the parameters of the model and test the statistical significance of those estimates.

Among the important methods used in the estimation of economic models (econometric estimating methods) and described in the literature are:

- Ordinary least squares
- Two-stage least squares
- Limited information single equation
- Full information maximum likelihood
- Three -stage least squares
- Instrumental variables
- Autoregressive schemes
- Distributed lag operators

A permanent "model specification data base" will be generated after estimation. This file will serve many purposes: it will allow the management of the complete model structure and its alternatives, and it will contain an input file for the simulation stage. This input file will contain all the necessary information such as: regression coefficients, sample endogenous and future exogenous data, constant adjustments, lags etc.. If this permanent data file is generated immediately after the model is estimated, the user can check his model, coefficients and residuals at any time without the problem of having the data bank updated before he checks his program.

Finally the model is simulated over a sample period or over any desired period. Since the models can be

simultaneous and non-linear, normally an iterative solution method is applied to the set of equations representing the model.

The user could experiment or represent policies for investigation either by supplying specific data or by carrying out special processes of analysis with the system by means of computations and transformations to the model under study.

The entire process is interactive in nature. It consists of small steps, each followed by decisions from the user on how to continue the process. This interaction between the user and the programs and data bases is represented in Figure 2.1 by a feedback loop.

A computerized system should provide the necessary programs to facilitate these interactive steps. The characteristics and facilities of a system to perform the above functions without undue restrictions is important to the success of the entire methodology.

2.4 Survey of Computer Modeling Systems

In view of the extensive number of computer modeling systems now available, a detailed comparison between the various languages and their capabilities is beyond the scope of this dissertation. However, this section will present some references to the literature which has attempted to do

so, and give a summary of their findings.

The first class of languages that should be mentioned in connection with programming of simulation experiments are the so called "general-purpose programming languages" such as FORTRAN, APL, ALGOL, PL/1, etc.. Although not specifically designed for any of the model building stages, they are still widely used to write special programs for estimation and/or simulation of each system under consideration.

The main advantage of using a general purpose programming language is of course its generality and flexibility. Basically the user can design an ad-hoc package for his model building requirements and data structures. However the principal shortcoming in using this approach lies in the inherent difficulties encountered both in writing and interpreting programs in these languages, as exemplified by the following excerpt:

"...the central difficulty of the problem is the control of the sequence in which the interdependent actions forming the model occur. If one attempts to write a simulation program using only a general-purpose language, one rapidly becomes enmeshed in the complexities of this sequencing control, which is not of great interest but nevertheless affords surprisingly fertile ground for minor errors. Moreover, mistakes here are liable to produce obscure effects, and are correspondingly difficult to eradicate." [WIL 64]

During the early sixties, a number of "simulation languages" were developed in order to facilitate the writing

of simulation programs for a variety of models and systems. Among the languages developed for simulation purposes, the following sample can be considered representative and include languages widely known and used:

- 1) GPSS [GOR 62]
- 2) SIMSCRIPT [MAR 62]
- 3) GASP [KIV 63]
- 4) DYNAMO [PUG 63]
- 5) SIMULATE [HOL 64]
- 6) CSMP [BRE,SIL 67]

Although these languages differ in intent and in the extent to which they can be applied to different modeling situations, their common objective has been to facilitate the representation of entities and events in a system.

In all the languages there are provisions for:

- Model initialization
- Maintenance of data files
- A Monitor or Executive routine which directs the flow and sequencing of activities in simular time (in case of discrete event simulation systems) in accordance with the language "world-view" (eg. events, entities, attributes).

There are also other aspects incorporated in the languages such as:

- Data collection facilities
- Report generation
- Statistical sampling procedures
- logical testing facilities
- Assistance for verification and validation analysis
- Assistance with debugging

The General Purpose Systems Simulator (GPSS) is a "block diagram" language where the structure of the system to be simulated is defined using a fixed set of predetermined block types. Each block will represent an action which can occur in the system under consideration. The sequence of actions are indicated by connections between the blocks of the diagram. The blocks are activated by the flow of basic units called transactions in simulated clock time.

SIMSCRIPT and GASP on the other hand are based on a description of the system in terms of concepts such as events, attributes, entities, set and state. In both languages the user provides the "event" routines, which are called at some system prescribed time. SIMSCRIPT is a procedural language in itself, while GASP can be considered a system of FORTRAN subroutines under the GASP EXECUTIVE which is also written in FORTRAN.

Although all these languages can be considered general purpose simulation languages, they differ mainly in the extent to which a particular simulation system can be represented with more or less difficulty, and their

associated solution procedures generated more or less automatically.

GPSS, SIMSCRIPT and GASP, although can be applied to a wide class of problems, are best suited to represent certain types of scheduling and queueing situations, or transaction oriented systems. DYNAMO and SIMULATE in contrast were specifically designed to handle large scale economic systems involving feedback mechanisms. SIMULATE solves nonlinear models by linearizing the equations using Taylor's series expansion, which leads to the Newton iterative solution algorithm. Program SIMULATE was written in FORTRAN for a CDC 1604 computer. It was successfully applied to the Klein-Golderberg econometric model of the United States. The analysis of models in terms of recursive blocks for efficient solution led Klein to the use of the Gauss-Seidel iterative method in connection with the Brookings-SSRC quarterly model of the United States. Linear subsystems of a model can be solved by matrix inversion, whereas iterative methods are used for nonlinear blocks of equations.

DYNAMO was developed in connection with the work of Forrester of MIT with the objective of making it easy for a user to represent a system and their interactions, as well as to help him in analyzing its behavior through computer simulation.

"Forrester's work led to a computer language called DYNAMO which allowed users without mathematical training to represent the system by a block diagram with boxes corresponding to flow rates and with other boxes modifying those flows rates -- and then to convert the diagram into a computer program." [ARB 77]

The Continuous System Modeling Program (CSMP) on the other hand was developed to model dynamic continuous systems (i.e., time is represented as a continuous variable) represented by means of differential equations. CSMP is a set of integration routines which permits the definition of a system in symbolic form in terms of differential equations. The language also accepts FORTRAN statements. It is a problem oriented language which frees the user from concerns regarding numerical integration etc..

Today some of these languages have extended their initial capabilities and evolved into very useful and sophisticated systems. For instance GASP can be used for hybrid simulation, with facilities provided for both discrete and continuous systems.

A summary of the capabilities of these languages and examples of their use in different modeling scenarios can be found in [NAY 68]. A comprehensive analysis and comparison between several simulation languages is provided by Teichroew and Lubin [TEI 66].

The systems discussed so far were designed primarily for the simulation and experimentation stages of model building.

Separate software was used for the analysis of the data and estimation of the parameters of the models. A number of systems were built for this particular phase, normally consisting of sets of statistical procedures written in a programming language such as FORTRAN. A recent survey and evaluation of this type of software used in the social sciences is given by Slys [SLY 74].

However, recent years have witnessed the development of a number of computer modeling systems designed more or less to encompass the whole of the model building activity, particularly in the social sciences. Some of those systems, used in econometric research, were surveyed by Prywes and this author as part of this dissertation research, and reported in [PRY 75b]. Perhaps the most representative and up-to-date system in this class, reflecting the trend in computer systems to aid the model building activity, is the TROLL system (Time-shared Reactive On-line Laboratory) developed by E. Kuh and M. Eisner at MIT and later continued as a project of the National Bureau of Economic Research (NBER) Computer Research Center for quantitative research in economics and other social sciences [TRO 73].

The TROLL system was born in 1966 as a project of the MIT Department of Economics. Its purpose was to bring "interactive programming technology to applied econometric research".

"The system was designed to give a researcher working at a computer terminal immediate, flexible access to all capabilities needed for entering, estimating, and simulating a model (equation system). Results are printed immediately on the user's terminal. This allow him to revise initial equations or execute alternative analytic procedures, as suggested by interim results. In short, the interactive design facilitates experimentation." [TRO 73]

Other state of the art computer modeling systems share some of TROLL characteristics and interactive approach (eg. PLANETS [BRA 72]), while others are batch oriented systems with more emphasis in the methods of analysis of time series data (eg. RAPE [RAD 72], TSP [TSP 73]).

Despite considerable progress attained to date by these systems in simplifying the interface between the model builder and the computer, they still have some inherent limitations and cannot be considered "automatic generators" of programs nor can they meet general users requirements because they have one or more of the following deficiencies:

- 1) They do not posses a generalized data structure.
- 2) They impose a particular "view of the world" to the user for representation of his system.
- 3) They require procedural knowledge to use, or
- 4) They require knowledge of a programming language such as FORTRAN for reasons such as inserting and writing subroutines.

Teichroew and Lubin recognized the need for more flexible data structures in their survey of simulation

languages when they stated:

"A language should provide for at least five types of variables: records, fields, groups of records, array or system variables, and lists of record identifications. More flexibility in creating and manipulating data structures is desirable because simulation models are becoming larger and more complex..." [TEI 66]

Chapter 3 of this dissertation summarizes the extensive capabilities of the MODEL language in this regard, and presents a simulation example in which structured data is manipulated and transformed in a non-procedural context.

Moreover, MODEL uses PL/1 as the target language for the generated programs, which was early recognized by the same authors to be a promising programming language in simulation applications:

"The new general purpose language, PL/1, may be particularly attractive for such applications because of its input-output features, its asynchronous operations, its flexible data structures, character and part-word data manipulating capability, and its list processing and memory extending commands."

The automatic generation of solution procedures techniques used by MODEL are described in chapter 6.

It is expected that the new approach of using automatic program generation techniques in model building, simulation and forecasting, will be particularly effective in situations where the dynamics of the problem require quick answers and rapid implementation, where the complexity of

the models make standard software development techniques impractical and cost-ineffective, where the data and/or models are continually changing, come from different sources and possibly require integration. These type of systems are becoming more common today and are recognized as part of a class of systems known as "decision support systems". Donovan [DON 77] in his study of database technology in which these systems are supported, recognized that the traditional approach is not adequate in itself for such systems, and recommended to develop new complementary technologies and the extension of existing database systems technologies for the development of effective decision support systems. He illustrates this point with the development and implementation of a Generalized Management Information System and with its application to a complex energy problem facing New England. The automatic programming methodology reported in this dissertation, although of different scope, expects to contribute also in this area, in particular by automating the program design and implementation process for a broad class of applications.

CHAPTER 3

The MODEL Language

3.1 Introduction

This chapter explains the Module Description Language (MODEL) through the specification of a simple but comprehensive simulation example from the field of economics. This by no means implies a limitation in the scope of usage of the language, which is generally independent of the class of problems and expected to be used in a wide variety of applications, but it is more of a reflection on the orientation of this dissertation to readers with competence in the area of social or engineering sciences, but not particularly knowledgeable of computer programming and information sciences.

MODEL has evolved from research on Automatic Program Generation, conducted at the Moore School, University of Pennsylvania since 1973. Its original goals were defined to be the automation of the program design and implementation process of a business data processing information system. It was intended to be used by management, business or accounting specialists without particular programming training. Those objectives were exemplified in [RIN 76].

Similar considerations were later on extended to the area of Automatic Test Program Generation for electronic equipment [CHA 77, TIN 77]. The original objectives have

been augmented to include the design and generation of programs for modeling, simulation and forecasting in the social or engineering sciences. This effort was carried out in conjunction with a major project undertaking which enhance the language capabilities, incorporate more sophisticated sequencing and iteration analysis, provide for interactive user communication facilities and relax some restrictions previously imposed in the complexity of the allowed input data structures [SHA 78].

This first section provides an overview of the language, its characteristics and novel features, and ends with a general discussion on the requirements imposed on the prospective user. Section 3.2 gives a brief description of the logical structure of the language and its components.

Detailed documentation for each of the language elements is presented in sections 3.3 through 3.4, while building the specification needed to automatically generate a solution program for the dynamic simulation of an econometric model.

The chapter concludes with a general discussion on interactiveness, and its implications in different stages of model building.

3.1.1 General Background

By looking at model building as an information process, it is possible to abstract a set of basic functions which are common to other fields such as business data processing.

Like these other systems, models deal with real world objects or events (entities) whose attributes are represented in the system by data: quantitative or qualitative measures of value.

This data is structured and grouped into data bases, which act as repository for the aggregate information. Any system must then provide some tool which will allow the user to manipulate his data; to apply transformations or functions which in turn generate new data, or modify the existing one, or allow for the restructuring of the information organization.

Different applications however have wide disparity in terms of the complexity of their processes. The information system for a typical department store with a large number of charge account customers, extensive and diversified stock inventory and a number of point-of-sale terminals connected to a computer network, will require complex file and interfile structuring while the transformations required by accounting rules would generally be quite simple. A macroeconomic model, on the other hand, requires sophisticated estimation and solution techniques for large systems of nonlinear equations, which transforms or operates on a generally simple structured databank consisting of a collection of time series. As in any information system, the designer of a system for either all or selected stages of model building should also give careful consideration to requirements of information selection and presentation to

the user as well as to the concept of interactiveness to be discussed in section 3.5.

Traditionally model builders have relied on the use of high level procedural languages such as FORTRAN or PL/1 to develop the systems or 'packages' which accomplish different tasks at different stages of the process. These languages are characterized by the use of control statements such as GO TO, IF, WHILE and DO loops plus INPUT/OUTPUT read and write statements, together with assignment statements which require awareness of the concept of memory, hence careful consideration of the logical order of these statements for proper evaluation. Recently there have been a proliferation of interactive systems which provide a command language for either all or selected stages of the modeling process. Despite the advances attained by these systems in terms of user requirements, they still need some degree of control logic and memory assignment in composing the commands for a particular task. Their main drawback however lies in poor capabilities to handle different data structures and in providing generality outside the narrow scope of their original design considerations. Real life situations often provide new alternatives to processes, making it difficult to abstract a closed set of imperative language sentences, unless the universe of discourse is a relatively simple one.

3.1.2 Overview and Intent of the Language

MODEL is a high level non procedural language designed to be employed by a broad class of users, specialized in different application areas but not necessarily in the art of computer programming. The language accepts descriptions of input and output data structures, together with computational statements called assertions which define relationships (arithmetical or logical) between them or their component elements or variables. These assertions are not of the assignment type, but rather considered to be algebraic tautologies. The user needs not to concern himself about concepts of sequencing of statements in composing his requirements into an integral unit or module. For instance in specifying the individual equations which compose a macroeconomic model, it is not necessary to keep track of the recursive order or causality between elements or sectors of the model. The processor will determine such order automatically and will report back complete documentation on the ordering together with an efficient coding of a solution program. By relieving the user from thinking in terms of a process to be carried out or 'flow-charting' for careful timing of instructions, it makes the language not only easier to use, but also permits concurrency in system development by allowing information to originate from different groups of users that share common description of a database.

The language allows both humans and mechanical processors to communicate about information structures. This facility has particular relevance in applications where there is great amount of data exchange, from different originations. In certain model building applications, data often comes in different formats and structures. Since in MODEL the data description is independent from the assertions that manipulate it, and from any specific hardware configuration, it can be used as a standard language for exchange of information about models and their data.

Since MODEL does not require step by step prescriptions for particular computations, as other standard procedural programming languages do, it is closely related to the language of mathematics. It uses short and concise statements which can be considered as axioms of a formal system. Computing theory provides the methodology to translate a non-procedural specification into algorithms for generation of programs [LEA 74]. Since a non-procedural language uses the language of mathematics, it is possible to assert properties of program correctness directly without recurring to the treatment of programs as static objects and without requiring an inverse translation to a different language (for example from a procedural programming language to the language of propositional calculus) [ASH 77].

The final product of the MODEL processor is a program

written in PL/1 to perform the information system requirements as described by the MODEL composed statements given by the user. These statements are entered via a text editor into a MODEL statement data base where subsequent analyses is done by the processor. These analysis include the resolution of inconsistencies, ambiguities and incompleteness of the totality of stored description for a particular module. Some stages of the analysis will require user interaction in order to add, delete or modify some of the statements.

Figure 3.1 illustrate the basic concepts involved in the automatic program generation process.

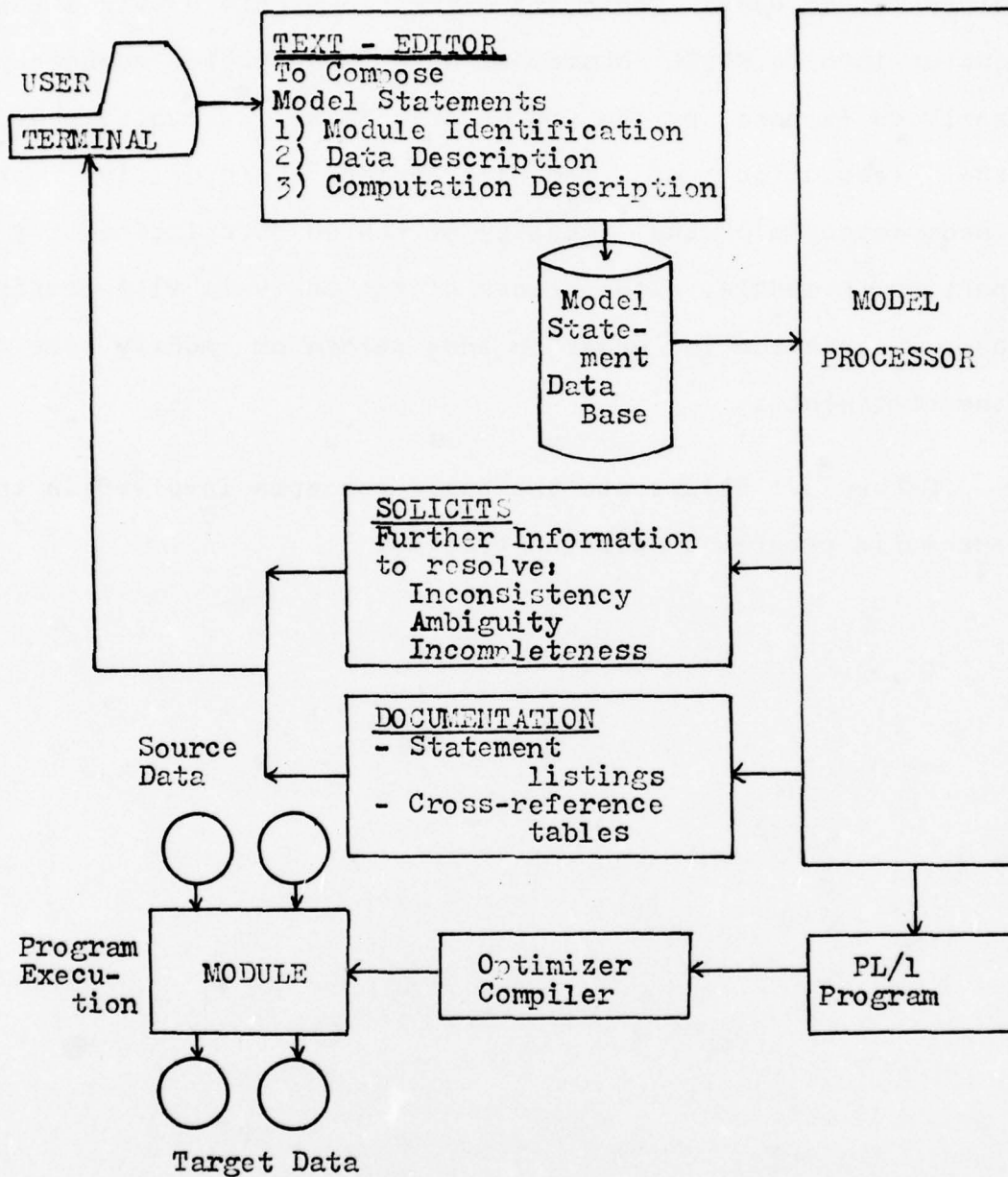


Figure 3.1

The Automatic Program Generation Process

Complete documentation is produced at different stages of the process together with any possible logical implication which may result from the analysis of the complete specification.

3.1.3 Main Characteristics and Capabilities

Following is a detailed description of the novel features which make the MODEL language unique in a variety of applications ranging from business data processing to simulation of engineering or economic systems. These are characteristics normally not found in other programming languages or special purpose systems used in model building:

1) Non-proceduralness - This concept was explained in section 3.1.2 and refers to the important feature of being capable of accepting unordered sets of assertions. This means that the user is not required (nor it is possible) to specify control logic for his program in terms of sequence of operations or memory assignment. This procedural information as well as the necessary input/output statements for the working of the generated program are to be deduced by the system.

2) Independence of Statements - A user can concentrate on composing one statement at a time. It is not necessary or possible to specify explicit relationships between statements. Implicit relationships like those holding among variables of a set of simultaneous equations as well as the recursive ordering of a model are done automatically by the system. An exception can be considered the case in which the user wants to 'force' a set of assertions to be treated by the processor as a simultaneous system of equations. This can be instructed by giving a common qualifier name to every assertion that belongs to the group. This subject is treated in more detail at the end of this chapter in section 3.5. However there is no loss of generality in the

language with this treatment. By definition simultaneous equations belong to the same equivalence class and are condensed by the MODEL processor into a compound assertion, representing a single statement with recursive interactions. This compound assertion is to be solved by an iterative method to be chosen by the user. If he gives the same qualifier name to a set of equations, the aggregate is treated as a single recursive statement for analysis, keeping this way the independence of statements characteristic of MODEL.

3) Randomness - This property is closely related to the concept of independence of statements previously described. It takes into account the fact that information may originate from a user or group of users at different times, without a well organized schema. Although an understanding of the input and output structures and their relationships is mandatory, the information about them can be entered at random, one statement at a time as information becomes available.

4) Incrementality - When the information provided by the user is incomplete or ambiguous, MODEL is capable of requesting additional information incrementally until a complete and unambiguous specification is given. This property is expected to give substantial savings in applications such as model building of large and complex systems. Traditional programming languages demand a great effort to keep track of proper control logic and to avoid costly testing and reprogramming for those systems.

5) Self-Documentation - This is generated at different levels throughout the interaction between the user and the processor and up to the flowcharting of the generated program. Cross references, summary tables and comments are generated by the processor. For example a user skillful in programming languages could use the flowchart and PL/I generated program documentation to modify it or to link it with other programs in a library. An economist on the other hand could use the cross reference table of name relationships to check instances of a particular variable in other sections.

6) Maintenance - Changes to programs either because of new system specifications or because of operational errors found can be implemented without recurring to an application programmer as a middleman. This will normally save time and avoids costly reprogramming due to misunderstandings.

7) Sharing - Sections of data or computation description included in the MODEL data base can be referenced by different users in order to incorporate them in their own programs, for instance the description of a databank containing time series for the United States economy. Then any program which wants to use this data needs only to refer to the previously stored description of it. If there are changes in the data structure, it is only necessary to modify the common description and automatically regenerate the programs that refer to it. Information sharing allows for exchange of knowledge and experience as well as of data from other users in similar application areas, as long as the description of their requirements is stored in the data base.

8) Integrability - Since the data descriptions are independent from the assertions that transform them, it is possible to decentralize the development of a system into independent module descriptions. These can be integrated later for a consistent overall specification. For instance, in developing a world trade simulation system, each national model could be specified following individual country standards and tested in isolation, later they can be integrated together into a world model which may include regional as well as country and commodity models by suitable inclusion of the linkage method specification. Chapter 7 illustrate this concept using a multi-country econometric model.

9) Tolerance - In composing the statements for a given application, many times it is found that the complete description requires repetitive and tedious detail. The processor will accept many types of omissions due to error or incompleteness and will provide automatically additional statements or will complete a partial specification of names in order to resolve ambiguities.

3.1.4 User Requirements

It has been stated that MODEL is a non procedural language which can be used as a program writer given the description of the information system needs. In this sense the duties of the MODEL processor are comparable to that of an application programmer. It was also mentioned that the user needs not to think in terms of processes, memory assignment or flowcharting in order to use the language to produce usable programs. Thus MODEL is oriented to application practitioners such as: business managers, scientists, economists, financial analysts, social researchers or accountants. In general people without exposure to computer programming concepts. The only requirement from this type of users, besides the assumed proficiency in their own area of competence and basic mathematical language, is a clear understanding of their information needs and in particular about the organization, both hierarchically and spatially, of the data structures that compose the system. These latter concepts relate to the identification of units and subunits composing the data (hierarchy of the organization structure) and to the order by which these subparts are arranged on a medium such as tape, disk or cards (spatial organization).

Since the data description facilities provided by MODEL closely resemble the declaration statements and data description sections of PL/I and COBOL respectively, a

proficient programmer could also benefit by taking advantage of existing file descriptions given in these languages.

In general the superiority of MODEL in terms of data independency and lack of control statements will provide any user with more time to dedicate to the analysis of information needs and structuring of the data.

3.2 The Language Structure

Figure 3.2 shows a breakdown of the different elements which compose the MODEL language. The user may provide a description for each of those elements in any order. Some omissions will be tolerated whenever the processor is able to provide the missing information in the completeness analysis. For instance the name of a module or the name of assertions will be provided by the system if not given by the user.

The header is composed of four subsections: (1) Module name used for identification purposes, names of files used as source (2) and target (3) in the specification of the module and references to other descriptions (called sections) previously stored in the data base (4). Those descriptions will automatically be included as part of the specification and may include computation description statements as well as data descriptions. This facility provides for the sharing of standard specifications previously entered into the system.

```

MODEL ---- HEADER ---- (1) MODULE NAME
STRUCTURE |             |-- (2) SOURCE FILE NAMES
           |             |-- (3) TARGET FILE NAMES
           |             |-- (4) REFERENCE TO SECTIONS
           |
           | DATA ----- (5) FILE -- FILE1 -- MEDIA
           | DESCRIPTION!  DESCR |         | DESCRIP
           |                 |         | - DATA
           |                 |         | DESCRIP
           |                 |         | - ASSERTIONS
           |                 |         | -LENGTH
           |                 |         | -EXIST
           |                 |         | -POINTER
           |
           |                 | FILEn -- MEDIA
           |                 |         | DESCRIP
           |                 |         | - DATA
           |                 |         | DESCRIP
           |                 |         | - ASSERTIONS
           |                 |         | -LENGTH
           |                 |         | -EXIST
           |                 |         | -POINTER
           |                 |
           |                 | - (6) INTERFILE
           |                 | POINTERS
           |
           | COMPUTATION --- (7) INTERIM VARIABLE DESCRIP
           | DESCRIPTION    | - (8) SUBSCRIPT PARAM DESCRIPTION
           |                 | - (9) ASSERTIONS -- IDENTITIES
           |                 | - BEHAVIORAL
           |                 | - EQUATIONS
           |                 | - OPERATING
           |                 | - CHARACTERIST
           |                 | - ETC..

```

Figure 3.2: Outline of MODEL Language

The data description section consist of two subsections: (5) File descriptions and (6) Interfile pointers to coordinate references in network like structures. The file description itself consist of the specification of the storage media, the description of the data constructs composing the file and their hierarchical and spatial organization and a set of assertions used to dynamically evaluate data dependent structures. These may include intra record coordinations (Pointer), variable length fields (Length) or assertions to compute the number of repetitions of certain structures (Exist).

Computational descriptions consists of defining those variables which are used in the computations but which do not appear as either source or target data names (Interim variables (7)). Variables in common use throughout the model as subscripts (ex. I,J,K,etc..) may also be declared as subscript variables (8). Finally the operating characteristics of the system under consideration should be inserted as assertions. In describing a model these will include definitional equations as well as stochastic and a priori equations composing the model, in addition to other auxiliary assertions needed for the working of the totality of the information system with its interactions.

3.3 An Illustrative Simulation Example

3.3.1 Introduction

MODEL can be used to automatically generate a program for the solution of a linear or nonlinear model whose relationships are expressed as assertions relating source to target elements. The assertions representing definitional, a priori and stochastic equations, plus the necessary relationships needed for the association between data structures, can be entered using the text editor in any order. The 'cycles' analysis phase of MODEL will first identify simultaneous equations (if any) and group them together into the model data base. Next, after completeness analysis, the 'sequencing' phase will order mechanically the assertions according to causality, for efficiency of the solution process in the generated program. A solution method* should be specified by the user whenever MODEL identifies groups of simultaneous equations, together with the associated parameters of the selected solution procedure (for instance initial values and convergence criteria at the variable level). For every group of simultaneous equations

* Presently only the Gauss-Seidel iterative technique for solution of nonlinear simultaneous equations is implemented. It is expected however that other methods such as: Newton's for nonlinear equations or Gaussian elimination for linear equations, will be added to the MODEL base.

in the description, MODEL will automatically generate the necessary code for the chosen method of solution. This process is transparent to the user. For instance the user can maintain the view of a language to state mathematical relationships, independently of processes, by observing that simultaneous equations are characterized by recursive references between data names (strongly connected components) for which it is possible to generate an iterative solution process whose convergence characteristics are determined by conditions stated in the contracting mapping theorem.

3.3.2 Description of Klein Model-I

A small Keynesian income-expenditure model of the U.S. economy by Klein [KLE 50] will be used to illustrate the specification of a simulation module for an estimated macro economic model. This model, known as Model-I, has been traditionally used both as tutorial and bench mark for testing new estimation techniques. Its purpose in this chapter, however, is to illustrate a different modeling process; that of formulating information requirements and structuring the data to satisfy those requirements. Doing so through MODEL seems an appropriate way of introducing the language features.

The model was estimated with annual data from 1921-1941. Aggregate consumption (C) is a function of current and lagged profits (P), and the sum of the wage bill in private industry (W1) and government (W2):

$$C = \alpha_0 + \alpha_1 W + \alpha_2 P + \alpha_3 P_{-1} + \hat{u}_1$$

$$W = W1 + W2$$

where* $\alpha_0 = 16.79$

$$\alpha_1 = 0.800$$

$$\alpha_2 = 0.020$$

$$\alpha_3 = 0.235$$

Net investment (I) is given in terms of current and lagged profits, and capital stock at the beginning of the year (K_{-1}).

$$I = \beta_0 + \beta_1 P + \beta_2 P_{-1} + \beta_3 K_{-1} + \hat{u}_2$$

where* $\beta_0 = 17.78$

$$\beta_1 = 0.231$$

$$\beta_2 = 0.546$$

$$\beta_3 = -0.146$$

Finally a labour demand equation gives the private wage bill as a function of national income (Y) plus business taxes (R) less the government wage bill (equal to private product (E)), both current and lagged, together with a time trend T1 measured in calendar years.

* Estimates reported are obtained by the method of full information maximum likelihood.

$$W1 = \gamma_0 + \gamma_1 E + \gamma_2 E_{-1} + \gamma_3^{66} (T1-1935) + \hat{u}_3$$

$$E = Y + R - W2$$

$$\text{where* } \gamma_0 = 1.600$$

$$\gamma_1 = 0.420$$

$$\gamma_2 = 0.164$$

$$\gamma_3 = 0.135$$

The identities closing the system are the expenditures and income definition of national income plus the definition of net investment:

$$Y + R = C + I + G$$

$$Y = W + P$$

$$I = K - K_{-1}$$

The endogenous variables are C, I, W1, Y, P, K, W and E and the exogenous variables: W2, R, G and T1.

Notice that in the specification of the Model-I equations, for simplicity the time subscript (t) has been left out from every description, but it is implicit in every variable (i.e., $I_t = K_t - K_{t-1}$ etc.). The MODEL specification presented later in section 3.4.4.4 will be structured in such a way as to closely resemble this notation.

3.3.3 Statement of the Problem

Figure 3.3 illustrates the basic layout for the simulation example of Klein's model I.

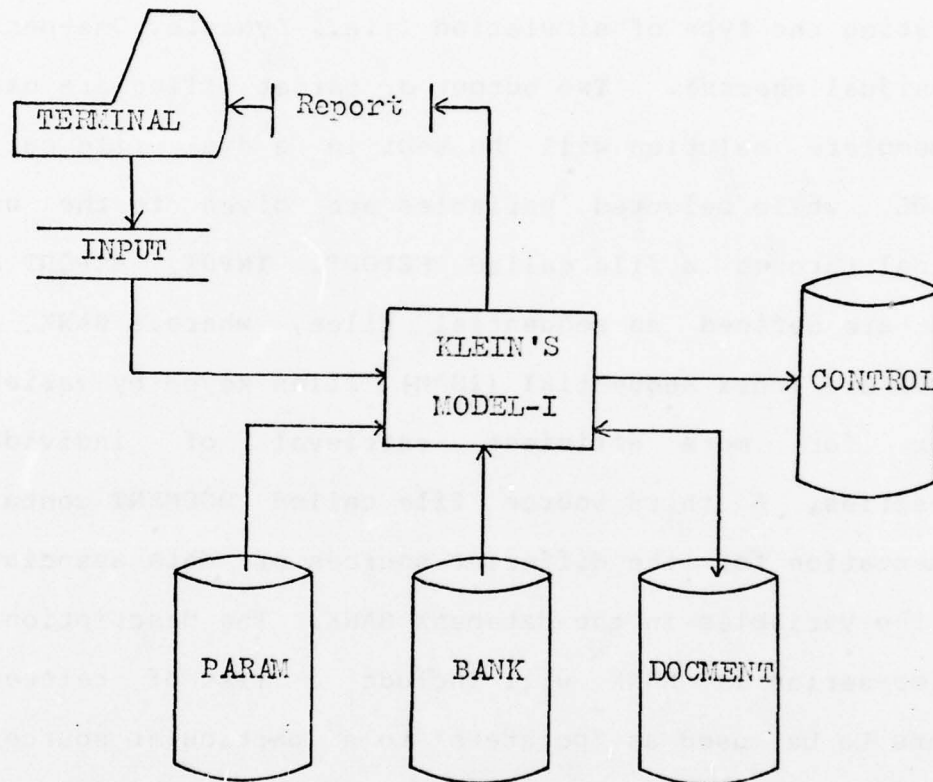


Figure 3.3
Diagram for Example
Klein's Model I

Data is read from file BANK and coefficients from file PARAM. The user will enter the required simulation parameters via the terminal. This source file is identified as INPUT and contains parameters such as: starting year for the simulation, number of periods to simulate and a code indicating the type of simulation (i.e., Dynamic, One-period or Residual checks). Two output or target files are used: the complete solution will be kept in a disk file called CONTROL, while selected variables are given to the user terminal through a file called REPORT. INPUT, REPORT and PARAM are defined as sequential files, whereas BANK and CONTROL are index sequential (ISAM) files keyed by variable number for more efficient retrieval of individual time-series. A third source file called DOCUMENT contains documentation for the different sources of data associated with the variables in the databank BANK. The description of a time-series in BANK will include a list of reference numbers to be used as "pointers" to a particular source of data. Therefore DOCUMENT is also an ISAM file with the data reference number used as key. Since many variables in BANK may share the same sources of data, this configuration will save the duplication of information which will occur if the sources of data were incorporated as part of each time-series.

A separate file (CONTROL) is kept for storing the solution, so that other programs could later access it and compare with the historical data in BANK for error analysis.

The solution file uses the same structure as BANK. By so doing it is possible to use CONTROL as input (source) file later, instead of BANK, to study "multiplier" effects. The "shocked" solution will then be stored on another file with the same structure as CONTROL for later comparisons.

The data description section of the example is given in a very general format, assuming the objective simulation program will be used by other models which may be annual or quarterly and with possible variable number of observations for the time-series in the databank. If the descriptions were limited to the solution of the model in this example only, with annual data from 1921 to 1941, 8 equations and 4 exogenous variables, it will be possible to substantially reduce the complexity of the given description.

3.3.4 Basic Concepts and Definitions

Information, in order to be usable for reference or computation, must be modelled into appropriate data structures. This modeling process involves the collection of data about objects or entities which are meaningful to the system, as well as providing the necessary descriptions for the repository of this data in terms of its logical structure.

The data description facilities of MODEL are used to provide a "template" of the hierarchical and spatial relationships among data instances. The following keywords

are used to compose data description statements expressing these relationships. The basic element of interest is the object or event occurring in the real world. This will be referred as a RECORD element. Associated with this entity there exists a set of attributes or basic items, be single units with an associated value, or a collection or group of items. Each of this items is called a FIELD. A FIELD is the basic unit of information and cannot be further divided into subunits. A third type of data description statement is the GROUP. This is an intermediate data structure used as generic collector of related RECORDs, FIELDs or other GROUPs.

Figure 3.4 shows a "tree" like structure depicting the hierarchical organization selected for each elementary record representing a time-series in the databank of the simulation example.

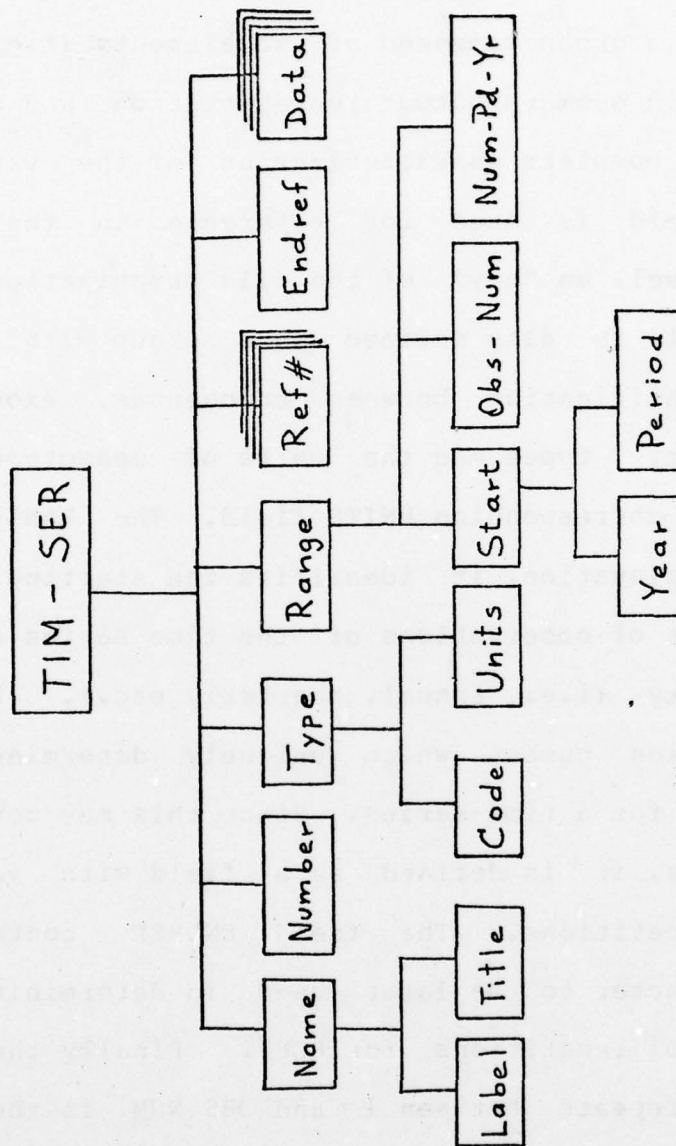


Figure 3.4

Hierarchical Model for Time-series Record

An explanation of the elements involved in this "model" for a time-series follows:

The NAME is a group composed of two elements (fields): a LABEL for use in summary output representation and a TITLE containing the complete characterization of the variable. The NUMBER field is used for reference in the model description as well as "key" of the file organization. The TYPE of variable is also defined as a group with a CODE entry for identification between endogenous, exogenous, definitional etc. types and the units of measurements is entered in the corresponding UNITS field. The RANGE group needs little explanation, it identifies the starting period and total number of observations of the time series as well as its periodicity (i.e., annual, quarterly etc.). REF# is an identification number which uniquely determines the sources of data for a time-series. Since this may come from several sources, it is defined as a field with variable number of repetitions. The field ENDREF contains a delimiter character to be later used in determining the actual number of repetitions for REF#. Finally the DATA entries which repeats between 1 and OBS_NUM is the last element of the data model.

A collection of these basic entities (RECORDs or GROUPs of RECORDs) is kept in a FILE, the largest aggregate of information in MODEL. There must be only one FILE associated with any RECORD (one to many relationship) and also one RECORD associated with any terminal unit or FIELD. Finally the direct "ancestor" or "parent" of a FILE is the MEDIA statement. This type refers to physically recognizable entities such as disks, tapes, terminals, cards, etc.

All these different data description statements can be depicted as a tree like hierarchical structure with the MEDIA type at the root of the tree and the FIELDS as terminal nodes. This representation is illustrated in Figure 3.5. The intermediate GROUP elements in brackets '{}' are optional, and can occur 0 or more times '*' (i.e., GROUP of GROUP's are possible).

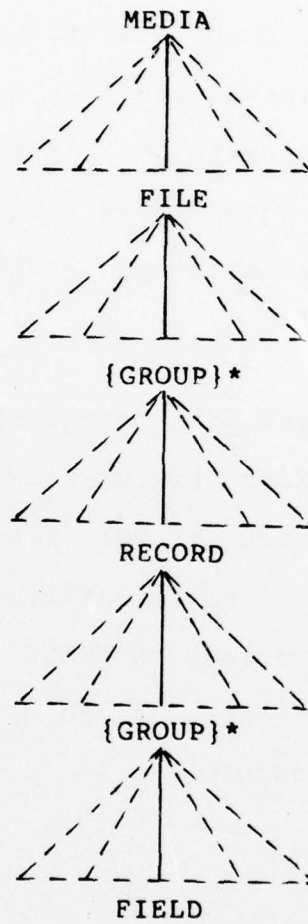


Figure 3.5
Hierarchical Data Description Statements in MODEL


```

| DISK IS MEDIA (UNIT=3330)
|--> BANK IS FILE (DISK)
|   |--> TIM SER IS RECORD (BANK,(*))
|       |--> NAME IS GROUP (TIM SER)
|           |--> LABEL IS FIELD (NAME)
|           |--> TITLE IS FIELD (NAME)
|       |--> NUMBER IS FIELD (TIM SER)
|       |--> TYPE IS GROUP (TIM SER)
|           |--> CODE IS FIELD (TYPE)
|           |--> UNITS IS FIELD (TYPE)
|       |--> RANGE IS GROUP (TIM SER)
|           |--> START IS GROUP (RANGE)
|               |--> YEAR IS FIELD (START)
|               |--> PERIOD IS FIELD (START)
|           |--> OBS_NUM IS FIELD (RANGE)
|           |--> NUM_PD_YR IS FIELD (RANGE)
|       |--> REF# IS FIELD (TIM SER, (1:20))----->
|       |--> ENDREF IS FIELD (TIM SER)
|       |--> DATA IS FIELD (TIM_SER, (OBS_NUM))
|
|--> PARAM IS FILE (DISK)
|   |--> COEFF IS GROUP (PARAM)
|       |--> CONSUMP IS RECORD (COEFF)
|           |--> ALPHA IS FIELD (CONSUMP, (4))
|       |--> INVESTM IS RECORD (COEFF)
|           |--> BETA IS FIELD (INVESTM, (4))
|       |--> WAGES IS RECORD (COEFF)
|           |--> GAMMA IS FIELD (WAGES, (4))
|
|--> DOCUMENT IS FILE (DISK)
|   |--> REFREC IS RECORD (DOCUMENT,(*))
|       |--> REF# IS FIELD (REFREC) <-----
|       |--> DESCRIPTION IS GROUP (REFREC)
|           |--> TITLE IS FIELD (DESCRIPTION)
|           |--> VOL# IS FIELD (DESCRIPTION)
|           |--> DATE IS FIELD (DESCRIPTION)
|           |--> PUBLISHER IS FIELD (DESCRIPTION)
|           |--> AUTHOR IS FIELD (DESCRIPTION)
|       |--> COMMENT (REFREC)
|
|--> CONTROL IS FILE (DISK)
|   |--> TIM_SER IS RECORD (CONTROL, (TOT_VAR))

```

Figure 3.6
Data Network of Source Files in Disk

AD-A066 192

MOORE SCHOOL OF ELECTRICAL ENGINEERING PHILADELPHIA P--ETC F/6 9/2
USE AND EXTENSION OF AN AUTOMATIC PROGRAM GENERATOR FOR MODEL B--ETC(U)
OCT 78 J L GANA

N00014-76-C-0416

NL

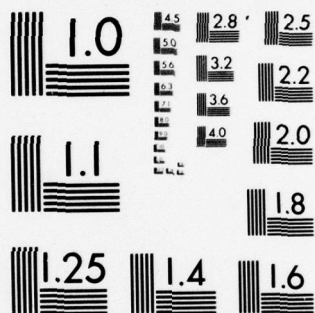
UNCLASSIFIED

78-02

2 OF 6

AD
A088162





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Figure 3.6 illustrate the network organization of all source files in a disk medium involved in the example. The hierarchical organization is given by indentation and by the arrows at the left of the diagram. The arrow at the right hand side indicates an inter-file relationship between the BANK and DOCUMENT files. The reference number key (REF#) in the TIM_SER record can be used to access the corresponding REFREC record in the DOCUMENT file (refer to section 3.3.4 for further details on interfile relationships).

Every node of the tree in Figure 3.4 is represented by a line in the specification corresponding to the BANK file in Figure 3.6. Also every node with the exception of the root of this tree like structure (the MEDIA declaration) is a subpart of another higher node. These relations between parents and descendants nodes are expressed in the language as an attribute of the data names. The parent name is written as the first element in a list of parameters enclosed in parenthesis for each declaration.

The files are progressively subdivided into records, groups and fields. Where data repeats a variable number of times, this is indicated by giving the minimum and maximum occurrences. For example the REF# field can repeat between 1 and 20 times (1:20). An assertion must be provided in the specification to compute the actual number of repetitions, unless this is given in the description as being the content of another source field. For instance the field DATA

occurrences is given by the content of the field OBS_NUM in the given specification. When the symbol (*) is used in a repeating item, it indicates that the actual number of occurrences is determined by the processor itself, using some physical delimiter such as end-of-file or end-of-record marks.

There may be other type of attributes associated with a particular data declaration. For instance the root or top element of the tree does not have a parent name as an attribute, however it contains such parameters as UNIT=3330 which identifies the physical device code for the random access disk. MODEL provides a number of parameters which are particular to specific declarations. These must be provided by the user. For instance the processor needs to know the type and length attributes associated with a particular field. If in the given example a LABEL consisting of a string of eight characters for a time series is to be stored in the group NAME, then it is necessary to specify it as follows:

LABEL IS FIELD (NAME,(CHAR(8)))

For a numeric field such as the starting year of the series the description could be entered as

YEAR IS FIELD (START,NUM(4))

In a FILE declaration it will be necessary to specify the access method to be employed with it. If the time

series are ordered by the field NUMBER, and every record can be accessed by this index, then it must be stated as follows:

BANK IS FILE (DISK,KEY=NUMBER)

There are default values provided by MODEL for most of this parameters. Others, if not given by the user, are deduced by the processor in accordance to the analysis of the totality of the statements.

The syntax and semantics specification for every component of the language is given in section 3.4 while illustrating the requirements for the simulation example of Klein's Model I.

3.3.5 Interfile Relationships

Many applications require to coordinate or express a relationship between occurrences of records in one file and instances of records in a different file. This is equivalent to a view of the data base as a network of inter-related files. This facility permits the partition of the data base into files containing only closely related data, and at the same time it avoids excessive duplication of common information between the files. If the value in the field of a given file uniquely determines or identifies a corresponding record in another file, then it is said that the field in the first file coordinates or "point" to the

record occurrence in the second file. In MODEL this coordinating rules are established by describing a POINTER type relationship. For instance in the data network shown in Figure 3.6, each entry of the field REF# in file BANK points to a particular record REFREC describing the sources of data in file DOCUMENT. Since different time-series may share a common documentation record, there exists a many-to-one relationship in this file structure. The content of each REFREC record contains information for the attributes associated with the origination of a given variable such as: date and title of the publication where the statistics were obtained for a sample period. The field REF# in the DOCUMENT file should also be declared as the "key" of the file organization as follows:

DOCUMENT IS FILE (DISK,KEY=REF#)

and the relationship between the records of file BANK and those of file DOCUMENT is expressed by the following declaration:

POINTER.DOCUMENT.REFREC = BANK.TIM_SER.REF#(FOR_EACH REF#)

This can be read as follows: each occurrence of field REF# in record TIM_SER of file BANK "points to" a record REFREC of file DOCUMENT. This process of concatenation using the (.) symbol from parent name to descendant name until a field is identified, is known as name qualifying. Qualified data names are used to distinguish between terminal nodes

which use the same data name. If the names are unique, there is no necessity for qualification, since this will be done automatically by the processor. POINTER is a reserved word in the language. It must be used as a qualifier for a pointer type relationship.

Since REF# is defined as a repeating data name, it is necessary to subscribe it with the expression "FOR_EACH REF#" to indicate all occurrences of the repeating field. Further explanation on the use of subscripts is given in the following sections.

3.3.6 Relating TARGET to SOURCE Data

In formulating a model it is necessary to express relationships holding between data coming into the system, called SOURCE data, and data to be produced by the system or TARGET data. These relationships are described with a notation similar to the one used in mathematical formulae or equations. Since they are used to map the content of fields from source files to the fields of target files by stating the actual mathematical relationship between them (directly or indirectly if INTERIM variables are used), they are referred as ASSERTIONS in MODEL language.

In the simulation example of Figure 3.1, the user will be required to relate the fields of the target files CONTROL and REPORT to fields in the source files INPUT, PARAM and BANK. The documentation file DOCUMENT can be used both as

source file, to obtain specific documentation for report generation, and as target file by generating a new REFREC in the module description. This new record will identify the values of endogenous variables in the CONTROL file as being generated by a "dynamic simulation" from the specification of Klein's Model I. Figure 3.7 shows some formulations using arithmetic expressions as well as free text in the form of 'string' constants.

Variable names which are used in the computational description, but which are not defined in the source or target data description, are called INTERIM variables and should be described as such. They can be used as synonyms of other data names or needed to express relationships between data in the files.

Target fields using the same name as corresponding source fields need not to be explicitly related by an assertion. By implication they are assumed equal. For instance the fields from the NAME group in BANK and CONTROL files, if not related by an assertion, are assumed to contain the same data by the processor.

TITLE.YEAR = 'YEAR'

TITLE.PERIOD = 'PERIOD'

Figure 3.7a
Assertions for Title Heading

OUT_YEAR = SIM_YEAR + (NP + SIM_PD - 2) / NUM_PD_YR;

OUT_PD = MOD (NP + SIM_PD - 2 , NUM_PD_YR) + 1;

Figure 3.7b
Assertions to Compute Year and Period of
Simulation in Actual Period NP for File REPORT

INCOME:

$Y(T) = C(T) + I(T) + G(T) - R(T);$

Figure 3.7c
Income Identity in Terms of INTERIM Variables
Assertion is Subscripted with Time Variable T

3.3.7 Use of Subscripts: Iteration

The treatment of subscripts as entities which coordinates occurrences of repetitive associate data names, but which are free from control iterative statements like the DO loop found in most programming languages, is one of the powerful features of MODEL.

A repeating data name can be subscripted by a <subscript expression> or by a variable which is declared as a SUBSCRIPT type variable. A <subscript expression> is an arithmetic expression with a subscript as operand. A subscript can be an integer or a variable. If it is a variable, the subscript keyword FOR_EACH followed by a reference to a repeating predecessor data name can be used whenever all occurrences are implied. If the repeating data name itself repeat, the predecessor data name can be omitted. The symbol (*) can be used as synonym of FOR_EACH. Other subscript variables can be used if declared with SUBSCRIPT statement.

Figure 3.8 shows some examples of the use of subscripts in composing the specification for Model I. The first example (3.8a) illustrate four equivalent ways to refer to every occurrence of the field DATA in the first record of file BANK. In Figure 3.8b, NP is defined as a subscript ranging from 1 to SIM_NP, the total number of periods to simulate. T is in turn defined in terms of the subscript NP and used as time index in the assertion for the consumption

function.

Subscripts are associated with a parent repeating data structure. It is always advisable to identify the <parent-name> when specifying a subscript. This is done by providing the name of the associated parent as the first parameter. For instance if OUTPUT is the repeating data structure indexed by subscript NP, the complete declaration should be:

```
NP IS SUBSCRIPT (OUTPUT,1,SIM_NP,1);
```

The completeness analysis phase will nevertheless provide the appropriate <parent-name> to a free subscript if this is missing.

It is also possible to define a subscript variable inside the assertion itself by using the extended form of the assertion with <if-or-for-clause>. This implies that the subscripted variable scope is limited locally to that particular assertion. Figure 3.8c shows examples of the use of subscripts with this format to assign lagged historical data into interim variables.


```

BANK.TIM_SER(1).DATA(FOR_EACH DATA);
TIM_SER.DATA(1,FOR_EACH DATA);
BANK.DATA(1,*);
BANK.DATA(1);

```

Figure 3.8a
Equivalent References for a Repeating Field

```

NP IS SUBSCRIPT((1,SIM_NP));
T = NP + MAXLAG;
CONSUMPTION:

```

$$C(T) = \text{ALPHA}(1) + \text{ALPHA}(2) * W(T) \\ + \text{ALPHA}(3) * P(T) + \text{ALPHA}(4) * P(T-1);$$

Figure 3.8b
Consumption Function in Model I with Subscripts

```

FOR J = 1 TO MAXLAG LET E(J) = BANK.DATA(8,LAG(8)+J);
FOR J = 1 TO MAXLAG LET P(J) = BANK.DATA(9,LAG(9)+J);
FOR J = 1 TO MAXLAG LET K(J) = BANK.DATA(11,LAG(11)+J);

```

Figure 3.8c
Use of Subscripts in Extended Form of an Assertion

Iteration in the program will be implied by declaration of subscripts, but it is not explicitly indicated in the module specification by bounded nested DO loops.

Subscripts are used as an alternative to the use of POINTERS to coordinate data structures. They may be more natural when used to express relationships between source and target data because of their resemblance to normal mathematical notation. However when subscripts are used to coordinate instances of data descriptions in a module, they affect their independence of statements. As a consequence different modules which may want to share the same data description but the same coordination could not do so. POINTER assertions, on the other hand, are independent of data descriptions, and therefore preferred to express these coordinating rules.

3.3.8 Use of Functions

Functions can be used to pass a value that is a result of a computation. They can be referenced in any arithmetical or logical expression by its name followed by a list of arguments separated by commas and enclosed in parenthesis.

Functions are important elements in MODEL, since they are used to complete the language in applications which require specific procedural computations. Modeling in particular require specification of complicated statistical

and arithmetic formulae, as opposed to the simple relationships needed for some other data processing applications. In this sense functions provide a mechanism for extending the application knowledge data base of MODEL. They also serve as synonyms for conventional mathematical notation. For example the SUM function is similar to the Σ symbol used in mathematical notation to denote summation. This and other functions used in model building such as MINV for the inversion of matrices, TRANSP for the transposition of matrices and other general functions for the generation of random variates are described later in chapter 8 of this dissertation dealing with the use of MODEL in the estimation phase of a model.

All built in functions available in the PL/1 library can be employed in MODEL. These include the conventional arithmetic functions (ADD, ABS, MIN, MAX, MOD, etc.), standard mathematical functions (LOG, EXP, SIN, COS, etc.) and string handling functions (INDEX, SUBSTRING, etc.).

The user can always define new functions in PL/1 and add them to the library. Function names are reserved words of the language and should not be used to name data or variables. In case of ambiguities the user should identify the functions referred in an assertion by declaring a FUNCTION statement for the assertion with a list of function reference names used in that assertion.

In Figure 3.3 of the Data Network the record TIM_SER

repeats TOT_VAR times. Since this field is not declared as a source field, an assertion must be provided to compute the actual number of occurrences. A function COUNT is provided by MODEL which computes or "count" the number of records read of the given record name parameter:

```
TOT_VAR = COUNT (BANK.TIM_SER);
```

the assertion indicates that the number of records in file CONTROL repeats as many times as TIM_SER records exists in file BANK.

Figure 3.7b on the other hand shows the use of the PL/1 function "modulo" MOD(x1,x2) which returns the smallest positive value R, such that

$$(x1 - R)/x2 = n$$

where n is a positive integer.

After this introduction to the capabilities of the language and to the example presented in section 3.3, next section proceeds to the composition of the complete specification of the module.

3.4 Module Specification

This section describes the various statements in MODEL while composing the specification for the simulation example presented in section 3.3. A formal description of the syntax of the language in Extended Backus Normal Form (EBNF) is given in [SHA 78].

3.4.1 General

The statements describing the model can occur in any order. However, as shown in Figure 3.2, the different types of statements can be divided into three categories or "sections":

- (1) Header Definition Section;
- (2) Data Description Section; and
- (3) Computation Description Section.

3.4.1.1 Syntax Notation

In describing the syntax of MODEL statements below, a conventional notation similar to Extended Backus Normal Form (EBNF) will be followed. Capital letters will be used to denote MODEL vocabulary words or keywords of the language. These must be provided "as is" by the user. Names in lower case letters and enclosed in angle brackets < > refer to a generic class for which the user must substitute an specific name or value item. Square brackets [] will be used to

indicate that the enclosed portion is optional (0 or 1 occurrence). Square brackets followed by an asterisk []* indicates optional number of repetitions for an item (0 or more occurrences). Alternatives are indicated by the symbol "|" meaning "or". Two colons followed by an equal sign "::=" are used to denote the expansion of an element into its components and can be read as "is defined as".

3.4.1.2 Character Set

The character set of MODEL is the same as that of PL/1, consisting of 60 characters. These include 26 letters, 10 digits, 4 alphabetic characters and 20 special characters.

<letter> ::= A|B|C... ..|Y|Z

<digit> ::= 0|1|2... ..|8|9

<alpha-character> ::= @|_|#|\$

<special character> ::= blank|.|<|(|+|_|&|*|)|;
|!|-|/|,|&|>|?|:|'|=

The special character | is underlined to differentiate it from the metalanguage symbol used for alternatives.

3.4.1.3 Operators

The following symbols are used to denote operators:

<arithmetic-operators> ::= <plus>|<minus>|<multiplication>
|<division>|<exponentiation>

<plus> ::= +

<minus> ::= -

<multiplication> ::= *

<division> ::= /

<exponentiation> ::= **

<comparison-operators> ::= <gt>|<not-gt>|<gt-or-eq>
|<eq>|<not-eq>|<lt-or-eq>|<lt>|<not-lt>

<gt> ::= >

<not-gt> ::= >

<gt-or-eq> ::= >=

<eq> ::= =

<not-eq> ::= =

<lt-or-eq> ::= <=

<lt> ::= <

<not-lt> ::= <

<logical operators> ::= <not>|<and>|<or>

<not> ::=

<and> ::= &

<or> ::= |

<string-operator> ::= <concatenation>

<concatenation> ::= ||

<group-operators> ::= (|)

3.4.1.4 Names

Data or variable <name>s can consists of any combination of <letter>s, <digit>s or <alpha-character>s, but the first

character of a <name> must be a <letter> or <alpha-character>. The length must not exceed 31 characters.

3.4.1.5 Use of Blanks

One or more blanks can be used freely in the declarations as delimiters between words of the language.

3.4.1.6 Comments

Comments are permitted and can be inserted in place of any blank used as delimiter.

<comment> ::= /*<character-string>*/

<character-string> may contain any character with the exception of the */ combination, which is used to signal the end of the comment.

3.4.1.7 Constants

<string-constant> ::= '<character-string>'

<arithmetic-constant> ::= <integer>|<decimal>|<binary>

<integer> can be any combination of digits with values from 1 to 32767.

<decimal> and <binary> constants can be represented as either <fixed> point or <floating> point numbers.

<fixed> point data consists of one or more occurrences of <digit>s with an optional <decimal>/<binary> point. If

no point is given, it is assumed to the right of the least significant digit. A sign may optionally precede a <fixed> point constant. For <binary> <fixed> point data the letter B must follow immediately the constant, without intervening blanks

<floating> point constants consists of a field of digits followed by the letter E, followed by an optionally signed <decimal> <integer> exponent. The entire constant may be signed and the first field may contain a <decimal>/<binary> point. For the <binary> constant the exponent specifies a power of 2 and is immediately followed by the letter B.

examples of arithmetic constants:

38

-0.438

11101E5B

3.1416

-111.101B

3.4.2 Composing the Heading Definition Section

The header consist of four statements which provide identification information for a program module: 1) the module name statement, 2) the source files statement, 3) the target file statement and 4) the reference statement.

Following is a description of the syntax and semantics of each of these statements.

3.4.2.1 Module Name Statement

Purpose: Identifies The Module.

Syntax: MODULE: <name> [;]
 |<name> [IS] MODULE [;]

Semantics: <name> is assigned to the module.

3.4.2.2 Source Files Statement

Purpose: Identifies input or source files to the module.

Syntax: S[OU[RCE]] [FILE[S]] : <qname> [,<qname>]* [;]
 |<qname> [,<qname>]* [<is>] S[OU[RCE]] [FILE[S]] [;]

 |ARE
 <is> ::= IS
 | =

 <qname> ::= <name> [.<name>]

Semantics: List of <qname>s are source or input files to the module. If any <qname> in the list contains two <name>s, then the first one must be the module name in which the source file is described and the second <name> is the file name itself.

3.4.2.3 Target Files Statement

Purpose: Identifies output or target files of the module.

Syntax: T[AR[GET]] [FILE[S]] : <qname> [,<qname>]* [;]
 |<qname> [,<qname>]* [<is>] T[AR[GET]] [FILE[S]] [;]

where <qname> and <is> have the same syntax as in the source files statement.

Semantics: List of <qname>s are target or output files to the module. If any <qname> in the list contains two <name>s, then the first must be the module name in which the target file is described and the second <name> is the file name itself.

3.4.2.4 Reference Statement

Purpose: Identifies sections of data or computation description previously stored in a library in MODEL data base.

Syntax: REF[ER]: <qualified-name>[,<qualified-name>]* [;]
 <qualified-name> ::= <name>[.<name>]*

Semantics: Sections given by <qualified-name>s will be searched from the library and the descriptions available in it will be included in the module for reference. <qualified-name>s are used in this context to avoid ambiguity by preceeding a <name> with another <name> which is an ancestor (section) in the data hierarchy. The first <name> must be the name of the module in which the section was specified.

3.4.2.5 Header Definition for Model I

The header definitions in Figure 3.9 essentially describe the block diagram of Figure 3.3. This particular specification does not refer to any previously declared section, hence the REFER statement is omitted. The commented titles are used for purposes of documentation only and do not affect the specification of the module. The processor treat them as blanks.

```

/*****
/*
/*          KLEIN MODEL I SPECIFICATION          */
/*
/*
/*****

/*****
/*
/*          HEADER DESCRIPTION                    */
/*
/*
/*****

MODULE: MODEL_I;

SOURCE FILES: BANK,PARAM,INPUT,DOCMET;

TARGET FILES: REPORT,CONTROL,DOCMET;

```

Figure 3.9
Header Definition for Model I

3.4.3 Composing the Data Description Section

This section provides definitions for the statements which describe each file used in a given module and their components: records, groups, fields and associated assertions. The user is responsible to maintain the hierarchical and spatial organization of his information requirements. Network structures consisting of interfile and intrarecord associations are also considered.

3.4.3.1 Media Statement

Purpose: To describe the physical medium on which a file is stored together with the attributes normally required to generate the Job Control Language (JCL) for a particular hardware installation.

Syntax: <name> [, <name>]* [<is>] MED[IA]
 [(<argument> [, <argument>]*)] [;]

<argument> ::= <blocksize-spec>
 | <recordsize-spec>
 | <organization-spec>
 | <recfm-spec>
 | <unit-spec>
 | <disp-spec>
 | <tape-label-spec>
 | <parity-spec>
 | <charcode-spec>
 | <page-spec>

```

|<line-spec>
|<trks-spec>
|<density-specs>
|<int-label-spec>
|<tab-spec>

```

```

+
|BS
<blocksize-spec>::=[ BLOCKSIZE [<is>] ] <integer> [;]
|BLKSIZE
+

```

```

+
|RL
|LRECL
<recordsize-spec>::=[                [<is>] <integer> [;]
|RECORDSIZE
|RECSIZE
+

```

```

<organization-spec>::=[ORG[ANIZATION] [<is>] <org-type> [;]

```

```

+
|D[IRECT]
<org-type>::= INDEXED SEQUENTIAL|IS|ISAM
|S[EQ[UE]NTIAL]]|SAM
|I[NDEXED]
+

```

```

+
|RF
|U[NDEFINED]
|FIXED BLOCKED|FB
<recfm-spec>::=[ RECORDFORMAT [<is>] ] V[ARIABLE]|VB [;]
|RECFM
|F[IXED]
+
|VAR_SPANNED|VS
+

```

```

+
|SYSOUT
|PUNCH
|CARD
+
|UNIT
|TAPE
<unit-spec>::=[<is>] ] DISK [;]
|DEVICE
+
|TERMINAL
|3330
|3330-1
|3400-1
|2314
|2311
|2305
+

```

```

+
|SHR
|OLD
<disp-spec>::=[DISP[OSITION] [<is>]] < [;]
|NEW
|MOD
+

```

```

+
|NL
|SL
|NONE
<tape-label-spec>::= TAPE LABEL [<is>] IBM STD [;]
|TL[ĀBEL] |ANSI STD
+
|BLP
|BYPASS
+

```

```

<parity-spec>::=PAR[ITY] [<is>] ODD [;]
|EVEN
+

```

```

+
|CC
+
|EBCDIC
<charcode-spec>::= CHARCODE [<is>] BCD [;]
|CODE
+
|ASCII
+

```

```

<page-spec>::= P[AGE SIZE] [<is>] <integer> [;]
|LINES-PER-PAGE
+

```

```

+
|WIDTH
<line-spec>::= LINESIZE    [<is>] <integer> [;]
|PAGE_WIDTH
|LINE_WIDTH
+

+
|TRKS
+
<trks-spec>::= TRACKS    [<is>] <  |7
|NO_TRKS              |9
+
+

+
|200
|556
<density-spec>::=DEN[SITY] [<is>] <    [;]
|800
|1600
+

+
|EL
|VOLSER
<ext-label-spec>::= <    [<is>] <name> [;]
|VOLUME
|EXT_NAME
+

<tab-spec>::= TAB [<is>] [(<integer>[,<integer>]*)] [;]

```

Semantics: The list of <name>s are described as storage medium to the module. The associated <argument> list gives the physical characteristics of a particular storage medium. Many of these attributes are optional. For instance CARD and PUNCH units need no further <argument>. Other units require more information as follows:

UNIT = TAPE <tape-description>

|DISK <disk-description>

|TERMINAL <terminal-description>

|PRINT <print-description>

<disk-description>::=[<blocksize-spec>,<recordsize-spec>,
 <recfm-spec>] [,<organization-spec>]
 [,<disp-spec>] [,<int-label-spec>]
 [,<ext-label-spec>] [,<trks-spec>]

<tape-description>::=[<blocksize-spec>,<recordsize-spec>,
 <recfm-spec>] [,<disp-spec>]
 [,<int-label-spec>] [,<ext-label-spec>]
 [,<tape-label-spec>] [,<parity-spec>]
 [,<charcode-spec>] [,<density-spec>]

<terminal-description>::=[<page-spec>] [,<line-spec>]
 [,<tab-spec>]

<print-spec>::=[<page-spec>] [,<line-spec>] [,<tab-spec>]

PRINTER and PUNCH can be used only as target files.

Most of the information in the media statement is used to generate Job Control Language (JCL) for a particular installation. The user can refer to the PL/1 Reference Manual for further assistance in the semantics of some of the arguments.

3.4.3.2 File Statement

Purpose: To describe a file and its attributes

Syntax: <qname>[,<qname>] [<is>] FIL[E]
|REP[ORT]

[((<parent-name> [,<argument>]*)] [;]

<qname> and <is> have the same syntax as described in the source file statement section 3.4.2.2. The <argument> list is the same as described in the Media statement section 3.4.3.1, with the following additions:

<start-file-spec>::= FILE# [<is>] <integer> [;]
|START_FILE

<key-spec>::= KEY [<is>] <qualified-name> [;]

+
|FILE_NAME
<dsn-spec>::= DSN[AME] [<is>] <qualified-name>[((<member>))][;]
|DATASET
+

<member>::= <name>|+<integer>|-<integer>

<space-spec>::= SPACE [<is>] <space-pars> [;]

+
|<integer>
|TRK
<space-pars>::=(Track[S] [,<integer>]
|CYL[S]
|BLOCKS|BLK
+

[, [<integer>][, [<integer>]]] [[,]R[EL[EASE]]] [;]
|RLSE
+

Semantics: List of <qname>s are described as files to the module. <parent-name> is the <name> of a parent or next section higher in the hierarchy (i.e., the name of a Media statement for a file). <argument> <start-file-spec> indicates the proper position of a file in a sequential medium such as a tape. The <key-name> <argument> is used to indicate on which field the file is ordered.

In creating new files it is not necessary to provide additional <argument>s. MODEL will provide as default standard <argument>s. For existing files not using the normal default values, it is necessary to provide matching <argument>s including <dsn-spec> as well as <blocksize-spec>, <recfm-spec>, <recordsize-spec>, <disp-spec> etc.. The <space-spec> is needed when creating a new file and the user wants his own space specifications.

3.4.3.3 Record Statement

Purpose: To describe records, the basic unit or entity of information organization.

Syntax: <qualified-name> [<is>] REC[ORD]

[[<parent-name> [, <argument>]*]] [;]

<argument> ::= <occ>[:<occ>] [, <occ>[:<occ>]]*

<occ> ::= <integer>
 | <name>
 | *

<qualified-name> has the same syntax as described in the reference statement section 3.4.2.4

Semantics: <qualified-name> is the name of the record. The record is an entity which is physically delimited in the file by an end-of-record mark. The <parent-name> must be the name of a section one level higher in the hierarchy of the tree structured description (i.e., a File or Group statement). The list of <occ>urrences gives the number of repetitions for the record. The first <occ>urrence of a group separated by colons gives the minimum <occ>urrence for a particular dimension, whereas the <occ>urrence following the ':' symbol gives the maximum number. If only one <occ>urrence appears in a group, then it is assumed to represent the maximum <occ>urrence. The asterisk '*' symbol is used to indicate a variable number of repetitions whose end is signaled by a physical delimiter (i.e., and end-of-file delimiter).

3.4.3.4 Group Statement

Purpose: To identify the name of a section which can contain other groups and fields, or other groups and records.

Syntax: <qualified-name> [<is>] GRO[UP]
 |GRP

the items <qualified-name> and <argument> have the same syntax as the corresponding items in the Record statement.

Semantics: <qualified-name> is the name of a group.
<parent-name> must be the name of a section one level higher

in the hierarchy of the tree-structured description. It could be the name of a file, another group or a record. The semantics of the <argument> list is the same as that of the Record statement section 3.4.3.3.

3.4.3.5 Field Statement

Purpose: To name and identify the basic items or attributes which compose groups or records and to give the characteristics of the data which they contain.

Syntax: <qualified-name> [<is>] FIE[LD]
|FLD

```
[(<parent-name>[,<argument>]*)] [;]
```

$$\langle \text{argument} \rangle ::= (\langle \text{occ} \rangle [: \langle \text{occ} \rangle] [, \langle \text{occ} \rangle [: \langle \text{occ} \rangle]]^*) | \langle \text{field-desc} \rangle$$

```

+      |PIC[TURE]   '[<pic-code> {(<length>)}] *'
+      |FIXED
+      |DEC[IMAL][,FLOAT]
<field-desc> ::= CHA[R[ACTER]]           [(<length>[,<length>])]
+      |NUM[ERIC]
+      |BIN[ARY][,FLOAT]
+      |BIT
+

```

$$\langle \text{length} \rangle ::= \langle \text{qualified-name} \rangle | \langle \text{integer} \rangle | *$$

`<pic-code> ::= B | X | A | 9 | V | Z | $ | / | . | S | _ | + | -`

`<qualified-name>` as well as `<occ>urrence` have the same syntax as described in the Record Statement section 3.2.3.3.

Semantics: The <qualified-name> is identified as a field of <parent-name>. <parent-name> is the name of a group or

record from which the field is the immediate descendent. The <field-desc> is the same as the data types found in PL/1 or COBOL. For an existing file in any of these languages it is possible to copy the needed descriptons directly. The meaning of these data type attributes is explained below:

PICTURE represents picture data. It is particularly helpful in describing data to appear in reports. It allows the description of each digit or character in each position of the field as alphanumeric, numeric or as special editing character. A symbol should be entered for each character or digit position in the associated data field as follows:

X for any character

A for an alphabetic or blank character

9 for a decimal digit or a blank

V for a decimal point

Z for a conditional digit (leading zeros are replaced by blanks)

Period (.). slash (/) or blank, are editing characters that may be inserted. S, +, -, \$, _, are also editing characters that may be used in static or drifting manner. A drifting character is similar to a zero suppression character. Further explanations on the use of PIC characters can be found in the PL/1 Programmers Reference Manual.

CHARACTER data may have a constant or variable length.

The number of repetitions or <length> is given by a list of <occ>urrences.

NUMERIC data is similar to CHAR, but consisting of numerics only. It must be of constant length.

DECIMAL data must also be of constant length. Its associated attributes consists of the length of the field and optionally the position of the decimal point.

BINARY data must be of constant length and up to 31 binary digits.

BIT is similar to CHAR, only the data consists of binary digits. If it is a variable length field, this must be given by a list of <occ>urrences.

The FLOAT attribute for DECimal and BINary numbers indicates a floating point number. DEC FLOAT precision should not exceed 33 and the exponent cannot exceed 2 digits. BIN FLOAT precision should not exceed 109 (binary digits) and the exponent must be in the range -260 to +256

In FIXED data representation the decimal or binary point is in a fixed position and this is indicated by adding after the length (total number of digits) of the field a comma (,) followed by the number of digits to the right of the decimal or binary point.

3.4.3.6 Files Description for Model I

Figure 3.10 shows the description of file PARAM, which is defined as a sequential file kept on a disk media. The file contains the estimated coefficients for the three stochastic equations in Model I: Consumption function, Investment function and wages. Each of the equations uses four coefficients which are kept in repeating fields ALPHA, BETA and GAMMA respectively. The file is defined as a single group COEFF which is divided into three records, one for each stochastic equation.

```

/*****
/*
/*          FILES DESCRIPTION          */
/*
/*          *****/
DISK IS MEDIA (UNIT=3330);

/*****
/*
/*          DESCRIPTION OF PARAM FILE          */
/*
/*          *****/
PARAM IS FILE (DISK,SEQUENTIAL);

COEFF IS GROUP (PARAM);

CONSUMP IS RECORD (COEFF);

    ALPHA IS FIELD (CONSUMP,DEC(14,8),(4));

INVESTM IS RECORD (COEFF);

    BETA IS FIELD (INVESTM,DEC(14,8),(4));

WAGES IS RECORD (COEFF);

    GAMMA IS FIELD (WAGES,DEC(14,8),(4));

```

Figure 3.10
Description of PARAM File for Model I

Figures 3.11a and 3.11b shows the description of file BANK and a sample entry for a TIM_SER record respectively. Each of the components in the declaration for this file were explained in section 3.3.

Figure 3.12 illustrate the description of files CONTROL and the documentation file DOCUMENT. Since the target file CONTROL uses the same structure as the source file BANK, it is only necessary to declare the file description statement and use the previously defined name for the record (TIM_SER), this time related to the parent name CONTROL. This is sufficient to imply the same description as the one given to file BANK. The description of all files in a disk medium is now complete.

```

/*****
/*
/*          DESCRIPTION OF BANK FILE          */
/*
/*
*****/

```

```

BANK IS FILE (DISK,KEY=NUMBER);

TIM_SER IS RECORD (BANK,(*));

NAME IS GROUP (TIM_SER);

    LABEL IS FIELD (NAME,CHAR(4));

    TITLE IS FIELD (NAME,CHAR(40));

NUMBER IS FIELD (TIM_SER,NUM(4));

TYPE IS GROUP (TIM_SER);

    CODE IS FIELD (TYPE,CHAR(4));

    UNITS IS FIELD (TYPE,CHAR(20));

RANGE IS GROUP (TIM_SER);

    START IS GROUP (RANGE);

        YEAR IS FIELD (START,NUM(4));

        PERIOD IS FIELD (START,NUM(2));

        OBS_NUM IS FIELD (RANGE,NUM(3));

        NUM_PD_YR IS FIELD (RANGE,NUM(2));

REF# IS FIELD (TIM_SER,NUM(3),(1:20));

ENDREF IS FIELD (TIM_SER,CHAR(1));

DATA IS FIELD (TIM_SER,DEC(10,5),(OBS_NUM));

```

Figure 3.11a
Description of File Bank

/* BANK FILE IS DESCRIBED WITH A GENERAL FORMAT WHICH CAN BE SHARED BY OTHER MODULES TO DESCRIBE THEIR DATA (EX. ANNUAL AS WELL AS QUARTERLY DATA).

FOR INSTANCE FOR KLEIN'S MODEL I, WITH DATA FROM 1920 TO 1941, A TIME SERIES ENTRY WILL LOOK AS FOLLOWS:

(FIELD NAME)	(ENTRY)
NAME.LABEL	C
NAME.TITLE	CONSUMPTION
NUMBER	1
TYPE.CODE	ENDO
TYPE.UNITS	BILLIONS 1934 DOLLARS
START.YEAR	1920
START.PERIOD	1
OBS_NUM	22
NUM_PD_YR	1
REF#(1)	1
REF#(2)	2
REF#(3)	3
ENDREF	#
DATA(1)	39.8
DATA(2)	41.9
.	.
.	.
.	.
DATA(22)	69.7

*/

Figure 3.11b
Sample Entry for Time-series
Corresponding to First Variable: Consumption

```

/*****
/*
/*          DESCRIPTION OF DOCUMENT FILE          */
/*
/*          ****
/*****

```

DOCUMENT IS FILE (DISK,KEY=REF#);

REFREC IS RECORD (DOCUMENT,(*));

REF# IS FIELD (REFREC,NUM(3));

DESCRIPTION IS GROUP (REFREC);

TITLE IS FIELD (DESCRIPTION,CHAR(80));

VOL# IS FIELD (DESCRIPTION,CHAR(4));

DATE IS FIELD (DESCRIPTION,CHAR(13));

PUBLISHER IS FIELD (DESCRIPTION,CHAR(20));

AUTHOR IS FIELD (DESCRIPTION,CHAR(20));

COMMENT IS FIELD (REFREC,CHAR(80));

```

/*****
/*
/*          DESCRIPTION OF TARGET FILE CONTROL          */
/*
/*          ****
/*****

```

CONTROL IS FILE (DISK,KEY=NUMBER);

TIM_SER IS RECORD (CONTROL,(TOT_VAR));

/*THE DESCRIPTION OF TIM_SER IS ASSUMED IDENTICAL TO THAT
PREVIOUSLY DESCRIBED FOR BANK. */

Figure 3.12
Description of Documentation File DOCUMENT
and Target Solution File CONTROL

Two files are specified for the terminal unit; a source file INPUT containing control parameters for the simulation, and a target file REPORT which presents selected solution values to the user. The declarations for these files are depicted in Figure 3.13.

The INPUT file contains two records: CNTRL_PARAM and LISTREP. The first record contains fields with the starting year and period for the simulation (SIM_YEAR and SIM_PD), the total number of periods to simulate (SIM_NP) and a description of the type of simulation (i.e., dynamic, one-period or res-check). LISTREP is divided into two fields: TOT# which contains the total number of selected variables to be included in target file REPORT, and a repeating field VAR#, which contains the time-series numbers of those variables selected.

The REPORT file is defined as a repeating group (OUTPUT) containing two records: TITLE and ENTRY. The fields in TITLE represent heading information while the ENTRY fields contain the data for a particular year, period, label and solution value for a selected variable. Both records use the PICTURE attribute to describe its fields. Some assertions associated with the description of file REPORT are given in Figure 3.13b with some comments on its purposes.

```

/*****
/*
/*      DESCRIPTION OF FILES IN TERMINAL      */
/*
/*      *****/

```

```

/*****
/*
/*      DESCRIPTION OF SOURCE FILE INPUT      */
/*
/*      *****/

```

INPUT IS FILE (TERM);

CNTRL_PARAM IS RECORD (INPUT);

SIM_YEAR IS FIELD (CNTRL_PARAM,NUM(4));

SIM_PD IS FIELD (CNTRL_PARAM,NUM(2));

SIM_NP IS FIELD (CNTRL_PARAM,NUM(2));

SIM_TYPE IS FIELD (CNTRL_PARAM,CHAR(12));

LISTREP IS RECORD (INPUT);

TOT# IS FIELD (LISTREP,NUM(4));

VAR# IS FIELD (LISTREP,NUM(4),(TOT#));

```

/*****
/*
/*      DESCRIPTION OF TARGET FILE REPORT      */
/*
/*      *****/

```

REPORT IS FILE (TERM);

OUTPUT IS GROUP (REPORT,(SIM_NP));

TITLE IS RECORD (OUTPUT);

YEAR IS FIELD (TITLE,PIC'(4)A');

PERIOD IS FIELD (TITLE,PIC'BB(6)A');

LABEL IS FIELD (TITLE,PIC'BB(5)A');

VALUE IS FIELD (TITLE,PIC'BB(5)A');

Figure 3.13a
Description of Files in Terminal

ENTRY IS RECORD (OUTPUT,(TOT#);

OUT_YEAR IS FIELD (ENTRY,PIC'ZZZ9');

OUT_PERIOD IS FIELD (ENTRY,PIC'(6)BZ9');

OUT_LABEL IS FIELD (ENTRY,PIC'BBB(4)A');

OUT_VALUE IS FIELD (ENTRY,PIC'BBSZZZ9V999');

/* ASSERTIONS FOR TITLE HEADING */

YEAR = 'YEAR';

PERIOD = 'PERIOD';

LABEL = 'LABEL';

VALUE = 'VALUE';

/* THE OUTPUT REPORT FOR SELECTED VARIABLES WILL APPEAR
AS FOLLOWS IN THE TERMINAL: */

/*

YEAR__PERIOD__LABEL__VALUE

1921_____1_____K_____184.121

*/

/* THE ENTRY RECORD WILL REPEAT FOR EVERY SELECTED VARIABLE
GIVEN IN INPUT.LISTREP, AND THE OUTPUT GROUP FOR EVERY
PERIOD OF SOLUTION. */

Figure 3.13b
Assertions for Files in Terminal

3.4.3.7 Inter-File Relationships: POINTER type assertions

Purpose: To coordinate occurrences of repeating data structures between files.

Syntax: POINTER.<qualified-name> = <qualified-name> [;]

where <qualified-name> has the same syntax as defined in the Reference Statement section 3.4.2.4

Semantics: The least qualified <name> of the expression at the right-hand-side must be a field <name> which "points to" or is a "key" of a repeating data structure which is qualified by POINTER.

Example:

The following declarations represent interfile relationships for the specification of Model I:

```

/*****
/*
/*          INTERFILE RELATIONSHIPS          */
/*
/*          *****/
POINTER.CONTROL.TIM_SER = BANK.NUMBER;
POINTER.DOCUMENT.REFREC = CONTROL.REF#(FOR_EACH REF#);

```

3.4.3.8 Assertions for Variable Length: LEN-type Assertions

Purpose: To provide an expression that determines the length of variable length fields.

Syntax: LEN.<qualified-name> = (arith-expression) [;]

where the syntax of <arith-expression> is given in section

3.4.4.3.8 Assertion Statement.

Semantics: The least qualified <name> of <qualified-name> must be described as a field of variable length which is determined by the evaluation of <arith-expression>.

Example: If DOCUMENT.TITLE were described as a variable length field

TITLE IS FIELD (DESCRIPTION,CHAR(1:80));

and the actual length of the field were delimited by the semicolon character ';', then the following expression can be used

LEN.TITLE = DELIM(REFREC, ';');

where REFREC is the name of the record in which the field is described and DELIM is a built-in function which search for the delimiting character and returns the actual length of the field.

3.4.3.9 Assertions for Variable Repetitions:

EXIST-type Assertions

Purpose: To provide an expression that determines the number of occurrences of a repeating structure.

Syntax: EXIST.<qualified-name> = <arith-expression> [;]

where the syntax* of <arith-expression> is given in section 3.4.4.3.8 Assertion Statement.

Semantics: The actual occurrences of a repeating data construct are determined by the evaluation of

```

/* ASSERTION TO COMPUTE THE NUMBER OF REPETITIONS OF FIELD
REF# */
EXIST.BANK.REF# = DELIM(TIM_SER,'#')/3;
/*'DELIM' FUNCTION SCANS FOR '#' DELIMITER SYMBOL IN THE
RECORD TIM SER (FIELD ENDREF) */

```

The computation description section provides definitions for the declarations of variable types associated with the specification of relationships or transformations between data structures: interim variables and subscript parameters. It also describes the structure of the assertions, which are the basic statements used in expressing operational characteristics of a system or relationships between data elements.

Purpose: To describe fields that are neither in source nor in target files.

Syntax: <qualified-name>[,<qualified-name>]* [*<is>*] INT[ERIM]
 [[*(*[<parent-name>][,<argument>)*]] *<i>*;]

items <qualified-name> and <parent-name> have the same

syntax as the corresponding items in the Field statement section 3.4.3.5

Semantics: The <qualified-name> which must not be described in a source or target file can be used internally in the module for assertions. <parent-name> is used when a hierarchy of variables with a tree like structure defined by the <qualified-name> is needed. The <argument>s have the same semantics as in the Field statements.

Example: Figure 3.14 shows the definition of INTERIM variables used in Model I.

/* IN ORDER TO FACILITATE THE TRANSCRIPTION OF RELATIONS,
INTERIM VARIABLES ARE USED TO DESCRIBE THE MODEL WITH
APPROPRIATE MNEMONICS. */

```

/*****
/*
/*                               ENDOGENOUS
/*
/*                               */
/*****

```

C IS INTERIM (NUM(10),(NO)); /* CONSUMPTION */

I IS INTERIM (NUM(10),(NO)); /* NET INVESTMENT */

W1 IS INTERIM (NUM(10),(NO)); /* PRIVATE WAGE BILL */

```

/*****
/*
/*                               DEFINITIONAL
/*
/*                               */
/*****

```

E IS INTERIM (NUM(10),(NO)); /* PRIVATE PRODUCT */

P IS INTERIM (NUM(10),(NO)); /* BUSINESS PROFITS */

W IS INTERIM (NUM(10),(NO)); /* TOTAL WAGE BILL */

K IS INTERIM (NUM(10),(NO)); /* CAPITAL STOCK

AT END OF YEAR */

Y IS INTERIM (NUM(10),(NO)); /* NATIONAL INCOME */

```

/*****
/*
/*                               EXOGENOUS
/*
/*                               */
/*****

```

G IS INTERIM (NUM(10),(NO)); /* GOVERNMENT SPENDING */

R IS INTERIM (NUM(10),(NO)); /* BUSINESS TAXES */

T1 IS INTERIM (NUM(10),(NO)); /* TREND VARIABLE */

W2 IS INTERIM (NUM(10),(NO)); /* GOVERNMENTAL WAGE

BILL */

Figure 3.14
Use of Interim Variables in Model I

3.4.4.2 Subscript Parameter Description

Purpose: To identify or coordinate instances of multidimensional or repeating data structures.

Syntax: <name>[,<name>]* [<is>] SUB[SCRIPT]
 [([<parent-name>][,<argument>]*)] [;]
 <argument>::=<integer>[,<integer>][,<integer>] [;]

Semantics: The list of <name>s can be used as subscript symbols throughout the module. <parent-name> is the name of a repeating predecessor, or can also be the "own" data name. The item <integer> is used to declare the range and increment of the subscripted parameters. If only <integer> is declared, the range is assumed from 1 to <integer>. Two <integer>s imply a range from first <integer> to second <integer>. The third <integer>, when given, denotes the increment of the subscripted names. A default increment of 1 is assumed if not given.

Example: NP IS SUBSCRIPT (1,SIM_NP);

NP is described as a subscript ranging from 1 to SIM_NP, the total number of periods to simulate. The increment is 1 by default.

3.4.4.3 Assertions

The assertion consists of a header, which includes a group of optative declarations: the name of the assertion, identification of source and target variables used in the

assertion, name of functions which are referenced and those declarations which are needed for the solution of simultaneous equations (initial values, solution method and convergence criteria). Following the heading declarations comes the assertion statement itself.

3.4.4.3.1 Assertion name

Purpose: To label the assertion.

Syntax: <assertion-name>::=<qualified-name>:

<qualified-name> has the same syntax as described in the Data Description Section 3.4.3.

Semantics: <qualified-name> is assigned as label to the assertion. If not given, the MODEL system provides a name.

Example: MODEL_I.CONSUMPTION:

3.4.4.3.2 Source Variable Description

Purpose: Identifies variables used as source in the assertion.

Syntax :<assertion-name> S[OURCE]: <variable>[,<variable>]*
<variable>::=<qualified-name>[<subscripts>]

and <subscripts> is defined in section 3.4.4.3.8 Assertion Statement.

Semantics: The list of variables are used as source or input variables to the assertion. If no source variables are given, then all the RHS variables in the assertion are

assumed to be source variables.

Example: MODEL_I.CONSUMPTION:

SOURCE: W,P,ALPHA

the variables W,P and ALPHA used in the assertion MODEL_I.CONSUMPTION are the source variables.

3.4.4.3.3 Target Variable Description

Purpose: Identifies variables used as target (dependent variables) in an assertion.

Syntax : <assertion-name> T[AR[GET]]: <variable>[,<variable>]
<variable> has the same syntax as described in Source Variable Description.

Semantics: The list of variables is used as target or dependent variables of the assertion. If no target variables are described for an assertion, then all the LHS variables in the assertion are assumed target variables.

Example: MODEL_I.CONSUMPTION:

TARGET: C

C is the target variable for the assertion MODEL_I.CONSUMPTION.

3.4.4.3.4 Function Reference Statement

Purpose: To identify names in an assertion or module specification which are function references.

Syntax: [<assertion-name>] FUN[CTION]: <list-of-functions>

`<list-of-functions> ::= <name>[(<arguments>)]
 [, <name>[(<arguments>)]]*`

`<argument> ::= <field-desc>[, <field-desc>]*`

where `<field-desc>` has the same syntax as the corresponding item in the FIELD statement, section 3.4.3.5.

Semantics: The names given in the `<list-of-functions>` are function references in the module specification (if no `<assertion-name>` is given) or in the named assertion. The `<arguments>` associated with a particular function name specify the attributes of the arguments in the function. These arguments are required if the function is in a library and the user arguments differ from those specified for the function.

Example: FUNCTION: MOD,COUNT

identifies the two functions used in the specification of Model I.

3.4.4.3.5 Solution Method Description

Syntax: `<assertion-name> SOL[UTION]:`

`<method>[, <arith-expression>]*`

```

      +
      | NEWTON
<method> ::= GAUSS_SEIDEL
      | G_S
      +

```

Semantics: The name of `<method>` identifies the solution procedure. It is envisioned that more procedures will be

available in the future. The list of <arith-expression> is used as parameters for the procedure. For instance the GAUSS_SEIDEL solution procedure uses the first <arith-expression> to determine the maximum number of iterations permitted.

Example: MODEL_I: SOLUTION: G_S,50

assertion MODEL_I (possibly) consisting of a group of simultaneous equations is to be solve by the GAUSS_SEIDEL iterative procedure with a maximum of 50 iterations.

3.4.4.3.6 Initial Value Description

Purpose: To give starting values to target variables of a simultaneous assertion.

Syntax: <assertion-name> I[NIT[IAL]]:

<simple-assertion>[,<simple-assertion>]*

<simple-assertion>::=<arith-expression>=<arith-expression>

the syntax of item <arith-expression> is given in section

3.4.4.3.8 Assertion Statement.

Semantics: INITIAL values needed for iterative solution methods are given by means of a list of <simple-assertion>s. Normally these will consist of equations assigning an <arith-expression> to a <variable> which is a target variable in the assertion.

Example: CONSUMPTION:

SOLUTION: G_S,50

INITIAL: $C(T) = C(T-1)$

the initial conditions for variable C in assertion CONSUMPTION is defined as its own value lagged one period.

3.4.4.3.7 Test Value Description

Purpose: To give a convergence criteria for target variables of a simultaneous assertion.

Syntax: <assertion-name> TEST:

<simple-assertion>[,<simple-assertion>]*

the item <simple-assertion> has the same syntax as the corresponding item in the Initial Value Description.

Semantics: Test values for absolute convergence criteria for iterative methods are assigned to target values by means of a list of <simple-assertions>. Normally these will consist of the assignment of an <arith-expression> to a <variable> which is a target variable in the assertion.

Example: CONSUMPTION:

SOLUTION: G_S,50

INITIAL: $C(T) = C(T-1)$

TEST: $C(T) = .01$

an absolute convergence criteria of .01 ($ABS(C(T) - C(T-1)) \leq .01$ for convergence) is assigned to variable C during period T.

3.4.4.3.8 Assertion Statement

Purpose: To define inter-file relationships. To define data

relationships or data transformations: operating characteristics, computation rules and decision rules. Also to supplement the data description with rules for computing variable length or repeating data structures. To select subsets of data.

Syntax: [<heading>] <assertion> [;]

<assertion>::= <if-or-for-clause> <assertion>

[ELSE <assertion>]

[<arith-expression>=<arith-expression>

<if-or-for-clause>::= IF <for-expression> [THEN]

|FOR

|LET

+

+

<for-expression>::=<simple-for>[& <for-expression>]

||
+

<simple-for>::=<arith-expression>[FROM <arith-expression>

|<rel-opr>

+

[TO <arith-expression> [BY <arith-expression>]]

|<rel-opr>

+

<rel-opr>::=<|<=>|>|=|=

<arith-expression>::=[<sign>]<relational>

<sign>::=+|-

<relational>::=<terms> [<rel-opr> <relational>]

<terms>::=<term> [<sign><terms>]

$\langle \text{term} \rangle ::= \langle \text{factor} \rangle [\langle \text{diadic-opr} \rangle \langle \text{term} \rangle]$

$\langle \text{factor} \rangle ::= \langle \text{primary} \rangle [\langle \text{exp-opr} \rangle \langle \text{factor} \rangle]$

$\langle \text{primary} \rangle ::=$ ($\langle \text{arith-expression} \rangle$)
 $\quad | \langle \text{arith-constant} \rangle$
 $\quad | \langle \text{string-constant} \rangle$
 $\quad | \langle \text{qualified-name} \rangle [\langle \text{subscripts} \rangle]$

$\langle \text{subscripts} \rangle ::= (\langle \text{arith-expression} \rangle [, \langle \text{arith-expression} \rangle]^*)$

$\langle \text{exp-opr} \rangle ::= * | _ | _ | _$

$\langle \text{diadic-opr} \rangle ::= * | /$

$\langle \text{heading} \rangle ::= \langle \text{assertion-name} \rangle | \langle \text{source-variable-description} \rangle$

$\quad | \langle \text{target-variable-description} \rangle |$
 $\quad | \langle \text{function-reference-description} \rangle$
 $\quad | \langle \text{solution-method-description} \rangle$
 $\quad | \langle \text{initial-value-description} \rangle$
 $\quad | \langle \text{test-value-description} \rangle$

the syntax of the $\langle \text{heading} \rangle$ components was given in sections 3.4.4.3.1 to 3.4.4.3.7.

Semantics: Conditional expressions at different recursive depth, as described in the syntax definition, can be performed. However for most applications it is not necessary to compose more complicate expressions than a first level conditional $\langle \text{if-or-for-clause} \rangle$. The equal sign '=' here means algebraic equality as opposed to the 'assignment' semantics used in other programming languages.

The construction of <arith-expression>s should be clear from the definition. Iteration in the program can be implied by the use of <if-or-for-clause>.

A number of assertions used as relationships between data structures, as well as to define the operating characteristics of Klein's Model I are given in the next section.

3.4.4.4 Computation Description in Model I

This section present a list of figures illustrating the use of assertions in composing the specification for Model I. Figure 3.15 explains graphically the meaning of the auxiliary assertions given in Figure 3.16. These assertions generalize the specification intended to be used in a simulation library, as opposed to a "special purpose" description just for Model I. The statements in Figure 3.17 are needed to relate predetermined data (exogenous and lagged data) to corresponding interim variables.

The core of Model I structure is given in Figure 3.18. By proper choice of naming conventions and use of interim variables the description becomes apparent to the reader. A user may want to change this portion of the specification to try new hypothesis formulation without the need to refer to other sections of the specification.

Figure 3.19 list the assertions required to relate the target files REPORT and CONTROL to the source files, sometimes via interim variables. Finally Figure 3.20 illustrate how to generate a new documentation record in file DOCUMENT to indicate that endogenous values in file CONTROL are originated from this particular solution specification.

/* SINCE THE RECORDS IN THE SOURCE FILE 'BANK' ARE OF (POSSIBLE) VARIABLE LENGTH, WITH (POSSIBLE) DIFFERENT STARTING PERIODS AND NUMBER OF OBSERVATIONS (REFER TO FIGURE 3.15), IT BECOMES NECESSARY TO COMPUTE THE STARTING PERIOD FOR THE SIMULATION (SOLUTION) AS AN OFFSET FROM THE FIRST OBSERVATION ENTRY. */

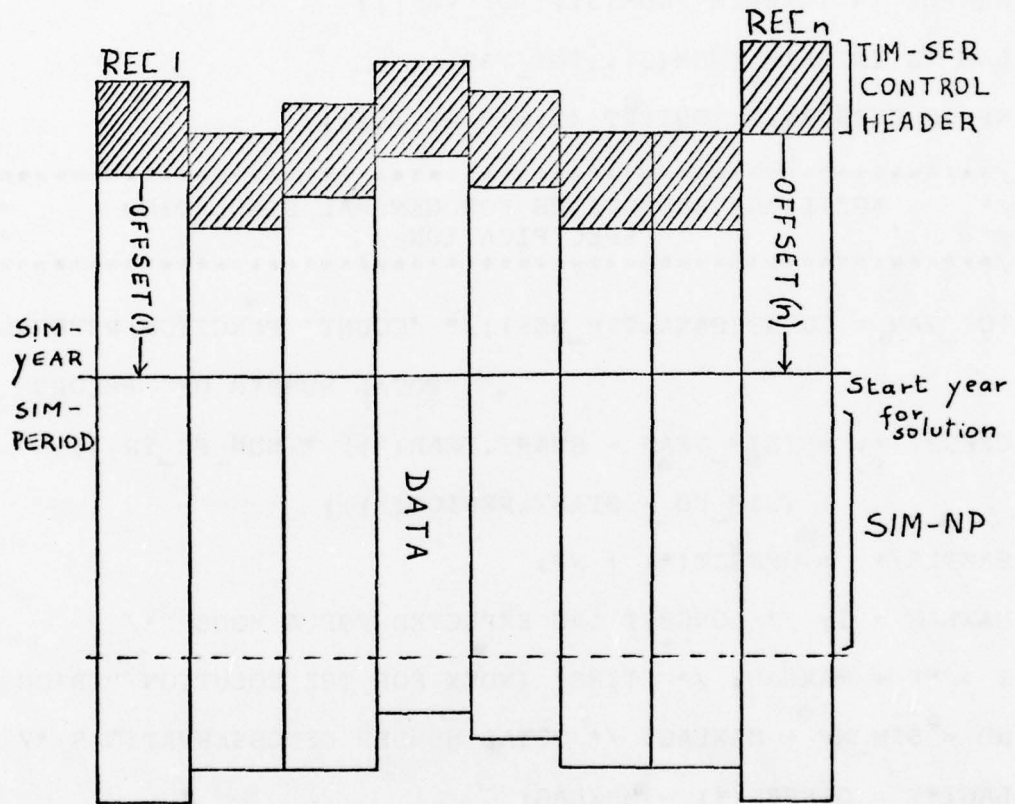


Figure 3.15
Diagram of BANK File Depicting
Computation of OFFSET for Solution

```

/* NEXT ASERTIONS COMPUTE THE  OFFSET VALUE FOR EACH RECORD
   AS WELL AS OTHER AUXILIARY PARAMETERS.  */

```

```

/*****
/*
/*          INTERNAL PARAMETER DEFINITIONS          */
/*
/*          *****/

```

```

OFFSET IS INTERIM (NUM(3),(TOT_VAR));

```

```

SAMPLE IS INTERIM (NUM(3),(TOT_VAR));

```

```

LAG IS INTERIM (NUM(3),(TOT_VAR));

```

```

NP IS SUBSCRIPT (OUTPUT,(1,SIM_NP,1,1));

```

```

/*****
/*          AUXILIARY ASSERTIONS FOR GENERAL SIMULATION          */
/*          SPECIFICATION                                          */
/*          *****/

```

```

TOT_VAR = COUNT(BANK.TIM_SER);/* 'COUNT' FUNCTION RETURNS
                                TOTAL NUMBER OF  RECORDS */

```

```

OFFSET(*) = (SIM_YEAR - START.YEAR(*)) * NUM_PD_YR(*)
            + (SIM_PD - START.PERIOD(*));

```

```

SAMPLE(*) = OFFSET(*) + NP;

```

```

MAXLAG = 1; /* LONGEST LAG EXPECTED FOR A MODEL */

```

```

T = NP + MAXLAG; /* 'TIME' INDEX FOR THE SOLUTION PERIOD */

```

```

NO = SIM_NP + MAXLAG; /* TOTAL NUMBER OF OBSERVATIONS */

```

```

LAG(*) = OFFSET(*) - MAXLAG;

```

Figure 3.16
Auxiliary Assertions for Generality of Module Specification


```

/* ASSIGNMENT OF EXOGENOUS VALUES TO INTERIM VARIABLES */
/* THIS PARTICULAR REPRESENTATION REQUIRES THE USER TO */
/* KNOW THE PHYSICAL LOCATION OF DATA (TIME-SERIES) IN */
/* THE 'BANK' FILE. */
VAR#1: G(T) = BANK.DATA(1,SAMPLE(1));
VAR#2: R(T) = BANK.DATA(2,SAMPLE(2));
VAR#3: T1(T)= BANK.DATA(3,SAMPLE(3));
VAR#4: W2(T)= BANK.DATA(4,SAMPLE(4));

/* OTHER ENDOGENOUS AND DEFINITIONAL VARIABLES ARE */
/* ASSIGNED AS FOLLOWS; */

/* C TO VAR#5 */
/* I TO VAR#6 */
/* W1 TO VAR#7 */
/* E TO VAR#8 */
/* P TO VAR#9 */
/* W TO VAR#10 */
/* K TO VAR#11 */
/* Y TO VAR#12 */

/* ASSIGNMENT OF LAGGED HISTORICAL DATA TO
INTERIM VARIABLES */

FOR J = 1 TO MAXLAG LET E(J) = BANK.DATA(8,LAG(8)+J);
FOR J = 1 TO MAXLAG LET P(J) = BANK.DATA(9,LAG(9)+J);
FOR J = 1 TO MAXLAG LET K(J) = BANK.DATA(11,LAG(11)+J);

```

Figure 3.17
Relating Interim Variables to Predetermined Data

```

/*****
/*
/*          MODEL_I EQUATIONS          */
/*
/*          *****/

```

INCOME:

$$Y(T) = C(T) + I(T) + G(T) - R(T);$$

PRIV_PROD:

$$E(T) = Y(T) + R(T) - W_2(T);$$

TOT_WAGE:

$$W(T) = W_1(T) + W_2(T);$$

PROFITS:

$$P(T) = Y(T) - W(T);$$

CONSUMPTION:

$$C(T) = \text{ALPHA}(1) + \text{ALPHA}(2) * W(T) \\ + \text{ALPHA}(3) * P(T) + \text{ALPHA}(4) * P(T-1);$$

INVESTMENT:

$$I(T) = \text{BETA}(1) + \text{BETA}(2) * P(T) \\ + \text{BETA}(3) * P(T-1) + \text{BETA}(4) * K(T-1);$$

PRIV_WAGE:

$$W_1(T) = \text{GAMMA}(1) + \text{GAMMA}(2) * E(T) \\ + \text{GAMMA}(3) * E(T-1) + \text{GAMMA}(4) * (T_1(T) - 1935);$$

CAPITAL:

$$K(T) = I(T) + K(T-1);$$

Figure 3.18
Model I Characteristic Equations

```

/*****
/*
/*          ASSERTIONS FOR FILE REPORT          */
/*
/*
*****/

N IS SUBSCRIPT (ENTRY, (1,TOT#,1,1));

OUT_YEAR(N,NP) = SIM_YEAR + (NP + SIM_PD - 2) / NUM_PD_YR;
OUT_PD(N,NP) = MOD (NP + SIM_PD - 2 , NUM_PD_YR) + 1;
OUT_LABEL(N,NP) = BANK.LABEL(VAR#(N));
OUT_VALUE(N,NP) = CONTROL.DATA(VAR#(N),T(NP));

/*****
/*
/*          ASSERTIONS FOR FILE CONTROL          */
/*
/*
*****/

CONTROL.START.YEAR(*) = SIM_YEAR;
CONTROL.START.PERIOD(*) = SIM_PD;
CONTROL.OBS_NUM(*) = SIM_NP;
CONTROL.DATA(1,*) = G(T);
CONTROL.DATA(2,*) = R(T);
CONTROL.DATA(3,*) = T1(T);
CONTROL.DATA(4,*) = W2(T);
CONTROL.DATA(5,*) = C(T);
CONTROL.DATA(6,*) = I(T);
CONTROL.DATA(7,*) = W1(T);
CONTROL.DATA(8,*) = E(T);
CONTROL.DATA(9,*) = P(T);
CONTROL.DATA(10,*) = W(T);
CONTROL.DATA(11,*) = K(T);
CONTROL.DATA(12,*) = Y(T);

```

Figure 3.19
Relating Target Data to Interim and Source Data

```
FOR J = 1 to 4 LET  
EXIST.CONTROL.REF# = EXIST.BANK.REF#;  
FOR J = 5 TO 12 LET EXIST.CONTROL.REF#(J) = 1;  
FOR J = 5 TO 12 LET CONTROL.REF#(J,1) = 99;  
/* RECORD 99 IN FILE DOCUMENT WILL BE SET TO INDICATE THAT  
   ENDOGENOUS VARIABLES ARE GENERATED FROM THIS PARTICULAR  
   SIMULATION RUN */  
DOCUMENT.REFREC.REF#(99) = 99;  
DOCUMENT.REFREC.TITLE(99) = 'MODEL_I DYNAMIC SIMULATION';  
DOCUMENT.REFREC.VOL#(99) = 'VOID';  
DOCUMENT.REFREC.DATE(99) = 'NOV-4-77';  
DOCUMENT.REFREC.PUBLISHFR(99) = 'VOID';  
DOCUMENT.REFREC.AUTHOR(99) = 'L.R.KLEIN';  
DOCUMENT.REFREC.COMMENT(99) = 'CONTROL SOLUTION GENERATED  
                                FROM SPECIFICATION IN MODEL LANGUAGE';
```

Figure 3.20
Control Solution Documented with new Record

The specification to generate a solution program for the dynamic simulation of Klein's Model I is now complete.

3.5 Alternatives for Module Specification

Several alternative specifications for the generation of a simulation program are possible. Without loss of generality the descriptions that follow assume every time-series in file BANK with the same number of observations OBS_NUM. Also the time subscript T is defined over the appropriate sample simulation period (it is always possible to map the source data into an standard target structure which satisfy these assumptions).

If the user prefers array notation instead of mnemonics, it is possible to express the relationships representing the model structure directly in terms of the data field names, or rename the data entries with interim arrays as depicted in Figure 3.21.

```
/*TWO INTERIM VARIABLES ARE DEFINED AND MADE EQUIVALENT TO
THE SOURCE AND TARGET DATA FIELDS*/
```

```
X IS INTERIM ( NUM(10), (TOT_VAR, OBS_NUM));
```

```
Y IS INTERIM ( NUM(10), (TOT_VAR, OBS_NUM));
```

```
X = BANK.DATA
```

```
Y = CONTROL.DATA
```

```

/*****
/*
/*          MODEL_I EQUATIONS          */
/*
/*          *****/

```

```
INCOME:
```

```
Y(12,T) = Y(5,T) + Y(6,T) + X(1,T) - X(2,T);
```

```
PRIV_PROD:
```

```
Y(8,T) = Y(12,T) + X(2,T) - Y(10,T)
```

```
TOT_WAGE:
```

```
Y(10,T) = Y(7,T) + X(4,T);
```

```
/* ETC. */
```

Figure 3.21

Module Specification Using Array Notation

3.5.1 One-period Simulations and Residual Checks

The description of the module have been given in terms of dynamic specifications for the solution of the model. For 'one-period' simulations it is necessary to substitute dynamic values used for lags (i.e., solution values of lagged periods) by corresponding historical source values. Residual checking on the other hand requires to substitute every endogenous (target) variable at the right hand side of an equation by its corresponding historical source value, solve the assertion algebraically and subtract the obtained target value from the source historical one.

Implementation of the above exercises could imply that every assertion should be entered more than once when describing the model structure as follows:

```

/*THE TYPE OF SIMULATION PARAMETER IS READ FROM TERMINAL*/
IF (DYNAMIC) THEN
    K(T) = I(T) + K(T-1);
IF (ONE_PERIOD) THEN
    K(T) = I(T) + BANK.DATA(11,T-1);
IF (RES_CHECK) THEN
    K(T) = BANK.DATA(6,T) + BANK.DATA(11,T-1);
IF (RES_CHECK) THEN
    RES(11,T) = BANK.DATA(11,T) - K(T);
    RES IS INTERIM (NUM(10),(TOT_VAR,OBS_NUM));
/*RESIDUALS WILL BE STORED IN MATRIX RES*/

```

The duplication of the given model structure could be avoided however by proper selection of interim variables, use of functions or a more convenient naming convention. For instance: if right hand side endogenous equations use the notation $Z(\langle \text{var-indexes} \rangle)$ while endogenous lags

$L(\langle \text{var-indexes} \rangle)$. Then the following declarations and assertions can represent the given structure:

```

IF (RES_CHECK | ONE_PERIOD) THEN L = BANK.DATA
      ELSE L = CONTROL.DATA;

IF (DYNAMIC | ONE_PERIOD) THEN Z = Y
      ELSE Z = BANK.DATA;

Z IS INTERIM (NUM(10), (TOT_VAR, OBS_NUM));
L IS INTERIM (NUM(10), (TOT_VAR, OBS_NUM));

CAPITAL:
      Y(11,T) = Z(6,T) + L(11,T-1);

INCOME:
      Y(12,T) = Z(5,T) + Z(6,T) + X(1,T) - X(2,T);
      /*ETC.*/

IF (RES_CHECK) THEN RES= Y - Z;

```

Many alternatives exists depending on user imagination and/or design objectives for the program. However it is important to avoid duplication of the model structure description not only because of savings in file space and effort from the user, but mainly because the nature of an exercise such as residual checking is precisely to check for errors in the description or specification of the model

structure. Clearly it is not appropriate to check for errors in a different equation (model) than the one actually used for the dynamic runs, unless some mechanical procedure is used to generate the duplicate structures (ex. the text editor).

At the conclusion chapter of this dissertation, a system is proposed which provides specific interfaces for individual applications. In implementing such a facility for econometric modeling, provisions should be made for three types of solutions: "dynamic simulation" in which the values for period "t" are taken from simulated values in "t-1", "single equation simulation", solving each equation independently for each variable, such as to track the characteristics of each equation and finally "one-period simulation" using historical data for endogenous lagged values. The interface should then generate the appropriate MODEL specification corresponding to the given equation structure.

3.5.2 Mechanisms for Dynamic Multipliers and Policy Analysis

For purposes of forecasting and policy making, it is important to analyze the dynamic properties of a model. It is always possible to obtain a new solution for every possible policy assumption (i.e., for every possible exogenous value representing a policy variable). However it becomes more convenient to use some quantitative notion of

the response of the model to specific policy changes. When a change in an endogenous variable is induced by a unit change in an exogenous variable, the ratio between the former and the latter is called the "multiplier". Knowing, for instance, the effect that an increase in 1 billion dollars in government spending has on the endogenous variables of a model, it can be used to assert the approximate impact that other expenditure changes will have on the economy. The basic notion of an "impact multiplier" which occur over a simple period of time can be extended to "dynamic multiplier" by considering the path of the endogenous variables over time, when the exogenous change or step is maintained beginning at period t and its effect is carried out through the system dynamics. More general multipliers can be defined by introducing combined exogenous policy changes at different periods, for instance a policy package consisting of both an increase in government expenditures and an increase in tax schedule.

To obtain these dynamic multipliers, it is necessary to simulate the models with different values for the policy variables and then compare its solution with a CONTROL solution, which uses actual values for exogenous variables in the sample past period.

By suitable specification of the module, MODEL can be used for these experiments. For instance in the specification of Klein's Model I, changes to exogenous

variables can be introduced from the terminal source file and assigned to interim variables with the following statements:

```
VAR#1: G(T) = BANK.DATA(1,SAMPLE(1)) + DELTA(1,T);
```

```
VAR#2: R(T) = BANK.DATA(2,SAMPLE(2)) + DELTA(2,T);
```

```
VAR#3: T1(T) = BANK.DATA(3,SAMPLE(3)) + DELTA(3,T);
```

```
VAR#4: W2(T) = BANK.DATA(4,SAMPLE(4)) + DELTA(4,T);
```

```
/*DELTA VALUES ORIGINALLY READ FROM SOURCE FILE  
TOGETHER WITH EXOGENOUS DATA */
```

The specification can be generalized to include "constant adjustments" in every structural equation as well as for all critical parameters in the model, this will give greater flexibility to the forecasters and policy makers, which can then cope with data revisions and policy changes without costly reestimation or reprogramming of the model.

Example:

CONSUMPTION:

```
C(T) = ALPHA(1) + ALPHA(2) * W(T) .  
+ ALPHA(3) * P(T) + ALPHA(4) * P(T-1) + CON(5,T);
```

the values of all CON adjustments are also specified as part of a source file. Changes can be introduced for specific time periods at the investigator discretion without the need to recompose the specification.

Once the user completes the composition of statements

for his requirements, he must submit them for execution by MODEL. At this stage a communication interaction starts where the processor will communicate back to the user documentation on his specifications (i.e., if cycles or simultaneous assertions are found, cross reference tables etc.) or report errors and warnings together with solicitation for more information in order to avoid ambiguity or for completeness. Basic concepts involved in this interactive process in relation to the simulation example are given in next section.

3.6 Interactiveness

The user can communicate interactively with the MODEL processor. This is important since it allows the user to compose, update or retrieve sections and sub-sections of the MODEL data base in a conversational mode. Interactiveness serve to guide the user in specifying his problem by supplying him with additional information on request and by providing feedback from errors found when composing his statements. The documentation normally generated by MODEL can be directed to a high-speed printer on request. It is always possible to execute every phase of MODEL in batch, however it is expected that the use of the interactive facility will reduce the time a user normally have to spend in preparing a module.

Another MODEL feature is its capability to generate programs which in turn are interactive. This is possible by proper selection of parameters in the MEDIA description and by describing an appropriate data structure which allows for such communication. In giving a summary of the modeling process in chapter two, it was stated that this was an interactive process in itself. This interactiveness was represented by a feedback loop between the data base and different stages in the process. In generating programs for such processes it will be of advantage to give considerations to conversational facilities.

3.6.1 The Text Editor

The present MODEL implementation utilizes OED under Time Sharing Option (TSO) [IBM 74] as a text editor to maintain the MODEL data base. TSO provides the necessary functions for storing, update or retrieval of sections or subsections in the data base as well as general data management facilities for interactive communication. In summary TSO interacts with the system through a command language which allows:

- Online data entry and retrieval
- Batch capabilities to submit jobs and route output to high speed printers
- Interactive programming and debugging

Presently TSO uses the Telecommunication Access Method (TCAM) of an OS/VS2 IBM environment.

3.6.2 The HELP Statement

HELP is a teaching aid provided during syntax analysis. It can be used to obtain a brief description of the syntax of any statement in the MODEL language, or to interpret the significance of an error code reported to the user. The HELP statement is not stored in the associative memory, as the other MODEL statements, but the requested information will be displayed immediately to the user. The following syntax conventions should be used with the HELP statement:

```

<help-statement>::= HELP [<argument>]
<argument>::= MODEL
              |SYNTAX
              |<statement-type>
              |<error-code>

```

If the argument is the keyword MODEL, then a short list of all allowed statements in MODEL is displayed.

The keyword SYNTAX should be used to obtain a summary of syntactical conventions used by MODEL statements.

The syntactical unit <statement-type> is used to obtain syntactical information on specific statement types. Presently the following keywords can be used:

<statement-type> describes the syntax of

MODULE	Module statement
SOURCEF	Source File Statement
TARGETF	Target File Statement
REFER	Refer Statement
MEDIA	Media Statement
FILE	File Statement
RECORD	Record Statement
GROUP	Group Statement
FIELD	Field Statement
INTERIM	Interim Statement
SUBSCRIPT	Subscript Statement
ASSERTIONS	Assertions

If an error code is given as argument, then the

corresponding description will be displayed.

HELP can be issued without arguments, in such case a brief description of the most recent error code is displayed, if one is found, else the argument MODEL is assumed as default.

3.6.3 Specifying Simultaneous Assertions

MODEL provides the capabilities to 'force' a set of assertions to be treated as a simultaneous set by giving a common qualifier name to every assertion that belongs to the group. This is common practice in some instances where a faster overall convergence is sought in solving a bigger set of simultaneous equations containing the 'forced' one.

If the user thinks that by inner iteration of this critical group overall speed of convergence can be improved, or if by any reason it is desired to impose a partition on a set of simultaneous assertions into smaller subsets (at any recursive level by proper selection of qualifier names), it is possible to override the grouping done by the processor by using this simple naming technique.

example:

COUNTRY.INNER1.PRICE:

SOLUTION: G_S,40 <assertion>;

COUNTRY.INNER1.CONSUMP:

. .
. .
. .

COUNTRY.ASS_NO_N:

SOLUTION: G_S,20 <assertion-n>;

```

--COUNTRY.<rest-of-qname>
|           1 to 20 iterations max.
|
|   --INNER1.<rest-of-qname>: <assertion>;
|   |           1 to 40 iterations max.
|   |
|   --end-of INNER1
|
|----end-of COUNTRY

```

Figure 3.22

Inner Iteration Description

All the assertions qualified by COUNTRY are treated as a simultaneous group containing a sub-group INNER1 which is inner iterated by the solution procedure as shown in Figure 3.22. For every iteration of the outer group, the inner group will iterate until convergence or up to the maximum number specified as parameter of the solution process.

3.6.4 Reporting of Simultaneous Assertions

After submission of a module for processing, the cycles analysis phase will search for loops and simultaneous equations. Circular definitions between elements other than assertions are normally reported as errors. However for cycles between variables referenced in assertions, with some trivial exceptions (ex. $A = A + 1$), it is not possible to differentiate between a user error in composing his description, from a model characterized by a simultaneous equation system. To resolve this problem MODEL rely on its interactive capability by always giving back to the user the identified groups of elements circularly defined, and requesting the proper parameters for solution or alternatively to open the loops by recomposing the assertions properly. As stated in section 3.1.3 this is consistent with the concept that elements which recursively reference between themselves are considered in the same equivalence class from a syntactical basis. Semantically however, not every recursive definition will converge to a solution by an iterative process.

Referring back to the simulation example in sections 3.3 - 3.4, after the model description is analyzed by MODEL, the following information is reported back:

THE FOLLOWING GROUP OF ITEMS ARE CIRCULARLY DESCRIBED

INCOME	TARGET IS Y
PRIV_PROD	TARGET IS E
TOT_WAGE	TARGET IS W
PROFITS	TARGET IS P
CONSUMPTION	TARGET IS C
INVESTMENT	TARGET IS I
PRIV_WAGE	TARGET IS W1

The assertion <heading> must then be given with parameters for the solution of this proper simultaneous equation model.

example:

```
INCOME:
SOLUTION: G_S,50
TEST Y = .1, C = .1 ;
```

since the assertions are grouped in the data base, it is possible to provide parameters using any of the assertion names. The solution procedure for the whole system is specified to be Gauss_Seidel with a maximum of 50 iterations. No initial conditions are declared, hence these will be the system default (i.e., the endogenous value lagged one period). The test value used to stop the iterations are the default of .01 for all variables with exception of Y and C, for which it is modified as indicated.

After analysis by the MODEL processor, a complete and formatted report is generated. This document includes the original source specification plus additional statements automatically generated by MODEL for completeness. The listing also includes documentation headers for each group of associated statements types such as Data Description, Assertions and File statements. Figure 3.23 shows a sample output from the specification of Model-I. The appropriate indentation in the text is also system generated for readability.

Simultaneous assertions are also grouped together in the output report for user convenience, as well as in the associative memory for later retrieval and generation of a solution procedure. Figure 3.24 illustrates the reporting of simultaneous equations from the example given in this chapter. A complete listing of the output reports produced by the processor from the specification of Model-I, including cross-reference and attribute report of data names as well as the complete precedence or adjacency list is presented in Appendix A.


```

/*****
/*
/*          MODEL_I MODULE DESCRIPTION
/*
/*****
MODULE: MODEL_I;
SOURCE FILES: INPUT,DCCPENT,PARAM;
TARGET FILES: REPORT,DCCPENT;

/*****
/*
/*          DATA DESCRIPTION STATEMENTS
/*
/*****
DISK IS MEDIA(UNIT=3330);
TERM IS MEDIA;
TERM IS MEDIA;
DISK IS MEDIA(UNIT=3330);

/*****
/*
/*          "REPORT" FILE DESCRIPTION
/*
/*****
REPORT IS FILE(TERM);
  OUTPUT IS GROUP(REPORT,(SIM_NP));
  TITLE IS RECORD(OUTPUT);
    YEAR IS FIELD(TITLE,PICTURE '(4)A');
    PERIOD IS FIELD(TITLE,PICTURE '80(6)A');
    LABEL IS FIELD(TITLE,PICTURE '03(5)A');
    VALUE IS FIELD(TITLE,PICTURE '03(5)A');
  ENTRY IS RECORD(OUTPUT,(TCT));
    CLT_YEAR IS FIELD(ENTRY,PICTURE 'ZZZ9');
    G IS FIELD(ENTRY,PICTURE '(6)029');
    C IS FIELD(ENTRY,PICTURE '00E(4)A');
    CLT_VALUE IS FIELD(ENTRY,PICTURE '00SZZZ9V999');

/*****
/*
/*          "INPUT" FILE DESCRIPTION
/*
/*****
INPUT IS FILE(TERM);
  CNTRL_PARAM IS RECORD(INPUT);
  SIM_YEAR IS FIELD(CNTRL_PARAM,NUMERIC(4));
  SIM_PD IS FIELD(CNTRL_PARAM,NUMERIC(2));
  SIM_NP IS FIELD(CNTRL_PARAM,NUMERIC(2));
  SIM_TYPE IS FIELD(CNTRL_PARAM,CHAR(12));
  LISTREP IS RECORD(INPUT);
  TCT# IS FIELD(LISTREP,NUMERIC(4));
  VAR# IS FIELD(LISTREP,(TCT#),NUMERIC(4));

/*****
/*
/*          "CONFOL" FILE DESCRIPTION
/*
/*****

```

Figure 3.23

Sample Formatted Report From Model-I Example

OUTPUT.PERIOD(N)=*PERIOD*;	00013176
SACC03:	00013177
CLTPUT.LABEL(N)=*LABEL*;	00013178
	00013179
	00013180
SACC04:	00013181
VALUE(N)=*VALUE*;	00013182
	00013183
SACC15:	00013184
OUT_YEAR(N,NP)=SIM_YEAR + (NP + SIM_PD - 2) /	00013185
DISK.BANK.TIM_SER.RANGE.NUM_PD_YR(N);	00013186
	00013187
SAU016:	00013188
O(N,NP)=MOD(NP + \$10004,DISK.BANK.TIM_SER.RANGE.NUM_PD_YR(N)) + 1;	00013189
	00013190
SACC17:	00013191
C(N,NP)=DISK.BANK.TIM_SER.NAME.LABEL(\$JCOL(N));	00013192
	00013193
SACC18:	00013194
OUT_VALUE(N,NP)=DISK.CONTROL.TIM_SER.DATA(\$10001(N),\$10002(NP));	00013195
	00013196
SACC05:	00013197
TCT_VAR=COUNT(EXIST.BANK.TIM_SER);	00013198
	00013199
SACC10:	00013200
NC=SIM_AP + MAXLAG;	00013201
	00013202
/*.....*/	00013203
/*	00013204
/* SIMULTANEOUS ASSERTIONS	00013205
/*	00013206
/*.....*/	00013207
	00013208
	00013209
PRIV_PROD:	00013210
E(NP,T(NP))=Y(NP,T(NP)) + K(NP,T(NP)) - W2(NP,T(NP));	00013211
	00013212
INVESTMENT:	00013213
I(NP,T(NP))=JETA(1) + BETA(2) * P(NP,T(NP)) + BETA(3) * P(NP,	00013214
\$10005(NP)) + BETA(4) * K(NP,\$10005(NP));	00013215
	00013216
PROFITS:	00013217
P(NP,T(NP))=Y(NP,T(NP)) - W(NP,T(NP));	00013218
	00013219
INCOME:	00013220
Y(NP,T(NP))=C(NP,T(NP)) + I(NP,T(NP)) + G(NP,T(NP)) - K(NP,T(NP));	00013221
	00013222
	00013223
CONSUMPTION:	00013224
C(NP,T(NP))=ALPHA(1) + ALPHA(2) * W(NP,T(NP)) + ALPHA(3) * P(NP,	00013225
T(NP)) + ALPHA(4) * P(NP,\$10005(NP));	00013226
	00013227
TOT_WAGE:	00013228
W(NP,T(NP))=W1(NP,T(NP)) + W2(NP,T(NP));	00013229
	00013230
PRIV_WAGE:	00013231
W1(NP,T(NP))=GAMMA(1) + GAMMA(2) * E(NP,T(NP)) + GAMMA(3) * E(00013232
NP,\$10005(NP)) + GAMMA(4) * (T1(NP,T(NP)) - 1935);	00013233
	00013234
/*.....*/	00013235
/*	00013236
/* END OF SIMULTANEOUS GROUP	00013237
/*	00013238

Figure 3.24

Automatic Grouping of Simultaneous Assertions in Model-I

CHAPTER 4

The MODEL Processor

4.1 Introduction

This chapter summarizes the mechanisms and procedures used by the MODEL system to automatically generate computer programs from a given non procedural user specification. The description presented here is based on a version presently under development by the Automatic Program Generation Project at the University of Pennsylvania (MODEL version III [PRY 77a,SHA 78,PNU 76,GAN 76]) and is a revision of the methodology previously used by [RIN 76] and extended later in version II of MODEL [PRY 77c].

Conceptually the processor can be considered as a function mapping the set of descriptive statements which conforms a particular module specification into a prescriptive flowchart and PL/1 object program which upon compilation and execution performs the desired application.

MODULE	PROCESSOR	PL/1 PROGRAM
SPECIFICATION	----->	FLOWCHART
(descriptive)		(prescriptive)

The transformation from a non procedural specification into a computational process is made possible by first translating the description into a directed graph representation where the nodes correspond to data items or

assertions stating relationships between them. The directed arcs connecting these nodes are inferred by the processor by analyzing the input statements for relationships holding between the nodes. Further analysis of this directed graph representation by means of some graph theoretical algorithms in conjunction with the interactive communication facility completes the translation into a consistent network which in turn can be interpreted to determine the sequence and control logic of the desired module.

Figure 4.1 shows a more refined view of the processor illustrating each of its phases as well as the feedback between them and the interactions with the user.

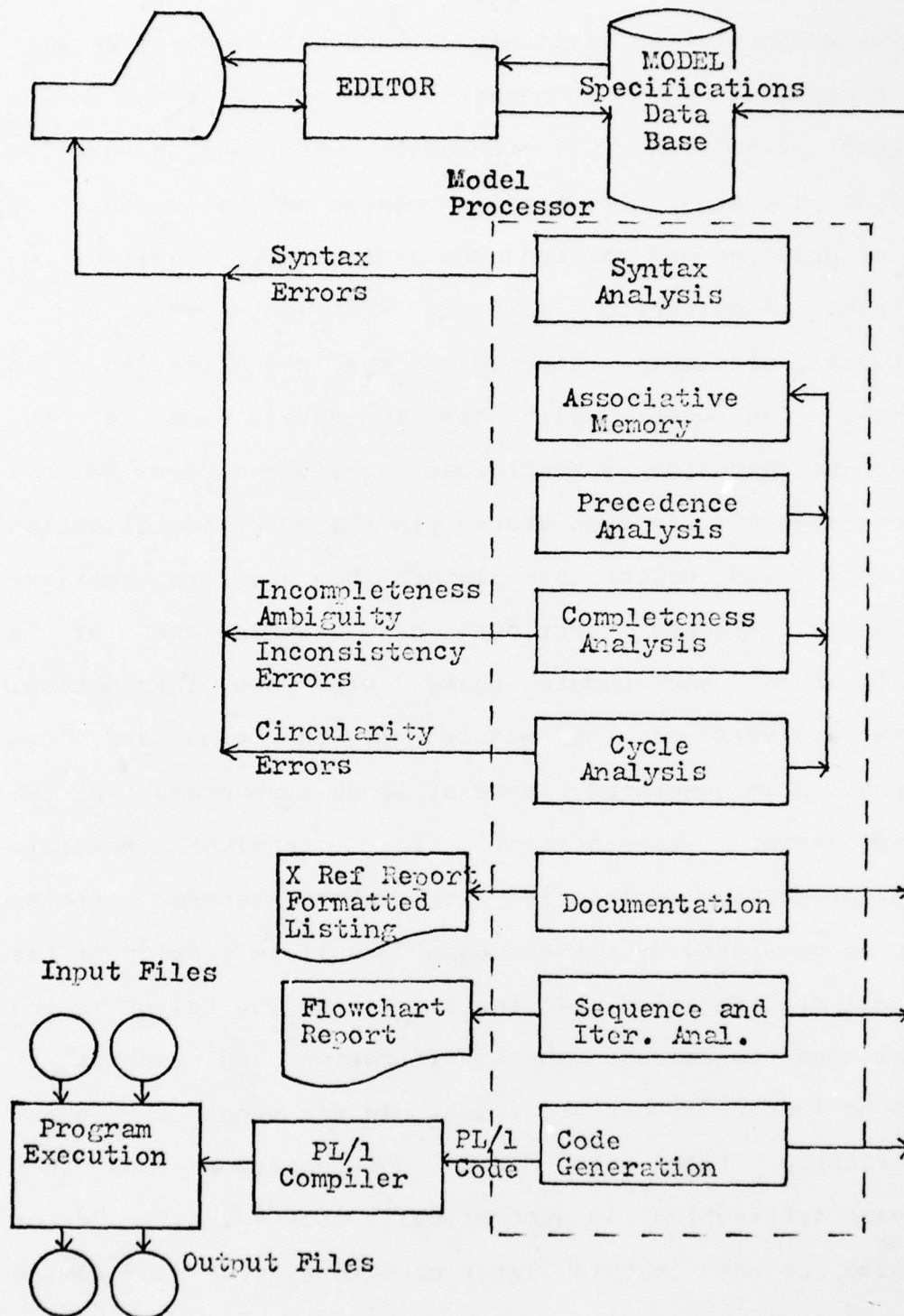


Figure 4.1
Overview of the MODEL Processor

The Editor is the main user communication interface with the processor. This component is external to the MODEL processor itself and was previously described in section 3.4.1 of chapter 3. Its function can be generally stated as that of creation and maintenance of the MODEL specification database. A collection of MODEL statements describing a functional module is referred as the specification of a program. The user will normally enter each of the statements composing a specification via the text Editor. These statements are then stored in the MODEL specification database. The editor can later be used to retrieve previously created sections or subsections of a specification, and update these with new information. During analysis of the module by the processor, new information is generated, some of it as a byproduct of the user-processor interactions to resolve possible inconsistencies, ambiguities or incompleteness errors. Other is generated by the processor itself to supplement the original specification. In the first case the Editor is the normal mean to augment the specification and resubmit it until no further errors are found. In the second case a new specification file with added documentation and other relevant information is automatically created. The Editor can also be used in this later case to examine and revise this newly created file, and if desired, substitute the original specification by the processor created one.

The next sections of this chapter describes each of the

components of the MODEL processor shown in Figure 4.1 in detail.

4.2 Syntax Analysis

This phase function is to check the local syntax and some semantics of the input specification. The front end Syntax Analysis Program (SAP) is in turn automatically generated by a Syntax Analysis Program Generator (SAPG), whose input is the syntax definition of MODEL in the Extended Backus Normal Form (EBNF) meta-language.

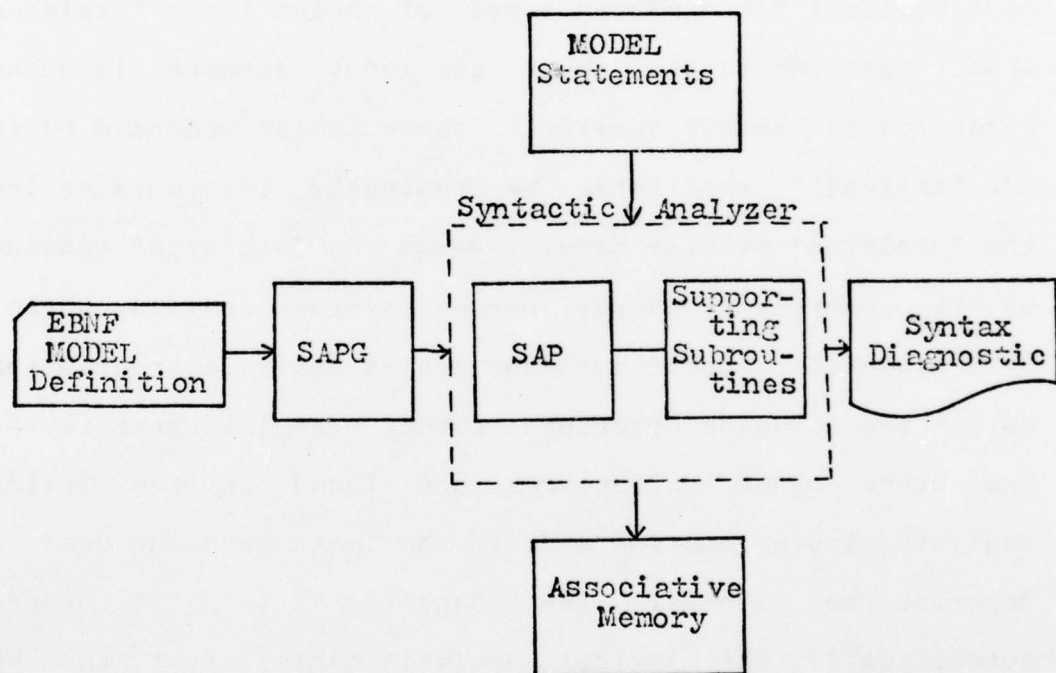


Figure 4.2

Block Diagram of the Syntax Analysis Phase

As shown in Figure 4.2, the automatically generated SAP is complemented with some hand-written supporting subroutines to be described later.

The SAPG was developed by the Data Description Language (DDL) project at the University of Pennsylvania, and its use and documentation has been described in detail in [RAM 73], [RIN 76] and [CHA 77].

The SAPG system uses techniques similar to those used by the well known XPL system [MCK 70], but with some noticeable distinctions; XPL produces a set of tables for a "skeleton" whose function is to check the input (source language) according to the BNF grammar. These tables produced by the XPL "analyzer" must later be physically incorporated into the "skeleton" program body. Hence the "skeleton" provided by XPL resemble a general purpose syntax analyzer whereas SAPG produces a specific ad-hoc syntax analysis program that parses the language described rather than interpret tables. Two other major differences are found in the lexical analysis implementation and in the meta-language used to describe the syntax. The "analyzer" in XPL deduces automatically the lexical analysis tables from the BNF grammar, whereas in SAPG a state of the art lexical analyzer based on finite state machine concepts (transition matrices) [CON 63] was implemented. Finally the specification of the MODEL language uses the Extended Backus Normal Form with Subroutine Calls (EBNF/WSC) as the metalanguage instead of

traditional BNF. This extension enables the SAP to check some of the statements semantics, to produce error messages and facilitates the encoding of source statements, yielding greater flexibility to the processor designer as well as providing for a better overall mechanism in posterior analysis and code generation.

Two general aspects of the SAP which must be mentioned in connection with the use of MODEL in model building applications are the following:

1) Local augmentation of the user description (statement) when needed. This feature is related to the concepts of tolerance of the language and completeness of the specifications. The following example illustrates how an incomplete description is extended as needed for subsequent phases after syntax analysis.

Assume an assertion stating a behavioral equation for a model of income determination entered as follows:

$$I(T) = A(1) + A(2) * R(T) + A(3) * (GNP(T-1) - GNP(T-2))$$

First SAP will provide a name for the assertion since this is not given by the user.

\$ASnnnn: <assertion body>;

Second the endogenous variable for investment (I) will be declared as "target" variable in the assertion since it appears at the left hand side of the '=' sign. Next the

data names at the right hand side of the '=' sign, namely interest rate (R) as well as the lagged variables for GNP will assumed "source" variables in the absence of other declarations. Notice that since SAP restrict itself to just local analysis, the fact that a variable is declared as "source" in a statement does not necessarily imply that it is an "exogenous" variable. The same variable may appear as target variable in other statement of the total specification. Subsequent phases will analyze globally the specification in order to assert possible interactions between endogenous variables as well as the proper recursive or causal ordering (refer to cycles and sequence analysis sections 4.5 and 4.8 respectively). Finally coefficients (A) will also be declared as source data names and the assertion after syntax analysis will look as follows:

SAS0001:

SOURCE: A(1),A(2),R(T),A(3),GNP(T-1),GNP(T-2)

TARGET: I(T)

$I(T) = A(1) + A(2) * R(T) + A(3) * (GNP(T-1) - GNP(T-2));$

Note that the same data names using different subscripts expressions are considered as different data items. Also any references to functions used in the statement will be provided as part of the assertion header if not given by the user.

2) "Merging" of simultaneous assertions. Whenever two or more assertions are given the same assertion name by the

user, the SAP will merge them by storing the set under a single storage entry corresponding to the common name. SAP also recognize as simultaneous any group of assertions which uses qualified assertion names and have a "simple name" in common. In this later case the different storage entries corresponding to each of the assertions using different qualified names are "linked" together in the associative memory (refer to section 5.3, chapter 5, for more detailed explanation on the treatment of simultaneous assertions).

Before describing in more detail some of the components of the syntax analysis phase, it must be emphasized that the use of this advanced "compiler-compiler" technology by MODEL to automatically generate the syntax analysis program has prove invaluable in that it allows changes to the language to be made relatively quickly, enabling rapid development changes and implementation.

4.2.1 The EBNF/WSC Metalanguage

The EBNF/WSC includes the conventional concepts of BNF, the traditional formal technique used to describe the syntax of mechanical languages [BAC 59] as exemplified in the ALGOL 60 report [NAU 60]. It consists of a series of "production rules" of the form "A::=S" where the metasymbols "::=" (two colons followed by equal sign) denotes a substitution rule for the non-terminal grammatical unit A from one or more alternative sequences of terminal or non-terminal units

denoted by S. Non-terminal units are enclosed in angle brackets "<>" whereas terminal units are written directly as sequences of characters in the object language vocabulary. The different alternatives are separated by the metasymbol "|" meaning "or". To facilitate the description of MODEL, BNF was extended first into EBNF by incorporating two more metasympols: square brackets "[]" used to represent optionality (0 or 1 occurrence) and square brackets followed by asterisk "[]" to indicate optional number of repetitions (0 or more occurrences). Next the EBNF description is augmented by including not only the syntax specification of MODEL, but also names of subroutines to be called after successful recognition of syntactical units (EBNF/WSC). The subroutines to be invoked are enclosed between slashes "/.../" and are inserted immediately following the appropriate syntactical unit. They may include the subroutines necessary for generating error messages, statement encoding routines, semantic checking routines and statement storing routines. A more detailed description of the EBNF/WSC characteristics and restrictions can be found in the references. The complete EBNF/WSC syntax definition of the present version of MODEL is given by [SHA 78].

4.2.2 SAP generation by SAPG

As indicated previously, the SAPG produces SAP given the description of the object language in EBNF/WSC. This is done in three passes over the set of production rules as

follows:

PASS1: As illustrated in Figure 4.3, the components of each production are scanned and sorted to form a set of tables. A "symbol table" is created with non-terminal units appearing at the left hand side of a production. These are the new production names. Non-terminals appearing at the right hand side of a production are included into a "work table" whereas terminal symbols will form a "terminal symbol table". Subroutine references are included in a separate table.

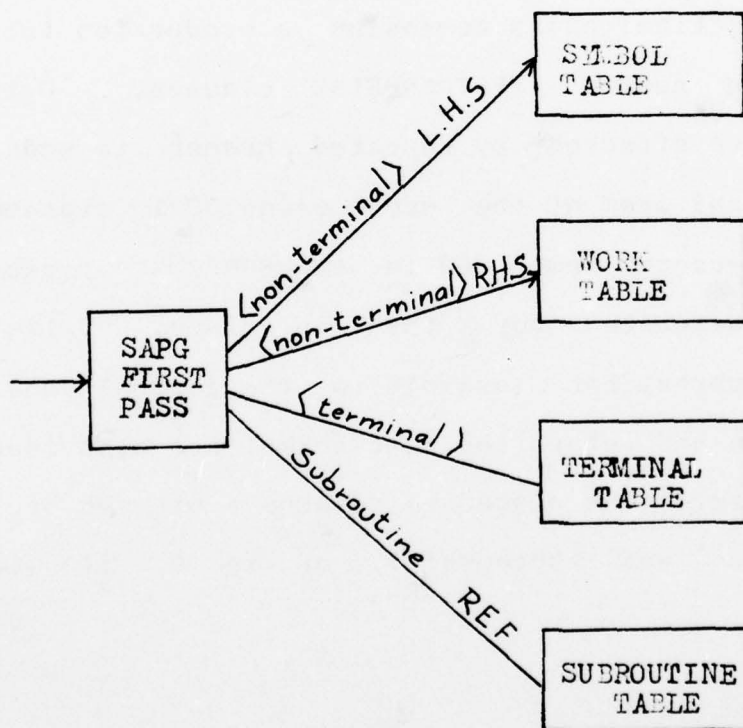


Figure 4.3

First Pass of SAPG

PASS2: This pass resolves the symbolic references in the work table. It checks that each right hand side non-terminal symbol in the work table is defined and links it to the corresponding entry in the symbol table. At this stage errors due to incomplete definitions and circularities in the specification are detected.

PASS3: This is the code generation phase and it is entered only if no errors were found in the preceding phases. The SAP is produced in PL/1. Each production rule from the EBNF/WSC specification will generate a PL/1 procedure. The exclusive syntactical units composing a production rule are scanned using nested IF-THEN-ELSE clauses. Optional occurrences are effected by repeated branch to scan the first syntactical item of the group using GO TO statements. Subroutine references embedded in the EBNF/WSC production get a CALL generated for them in place. Calls for housekeeping subroutines (example to the lexical analyzer LEX, which scan and return the next token) are also inserted in the procedure. The procedure returns a bit set to 1 if the recognition was successful, or to 0 if it was unsuccessful.

4.2.3 Supporting Subroutines

The hand-written supporting subroutines used in the syntax analysis phase can be one of the following types:

(1) A lexical analyzer (LEX) which scan for syntactic units or "tokens" and then returns to the SAP for syntactical checking. LEX works as a finite state machine based on the entries of a transition matrix. Rows and columns of this matrix represent states or character classes for the MODEL language. These classes divide the entire character set into categories such as: delimiters, numerals, illegal characters etc.. The class corresponding to a particular row represents the current state of the machine, while the column class represent the next state. The corresponding entry in the transition matrix represent an action to be taken, such as: concatenation of current token to next character, skipping of blank sequences or printing of an error message for an illegal character.

(2) Statement semantics checking routines; these routines are optional and are included at the discretion of the processor designer. They can be used to check if a range or condition in a syntactic unit is locally correct (for instance checking that the repetition factor of a field is between the integer bounds for the installation). Global analysis of the specification for overall consistency is performed in subsequent phases of the processor.

(3) Error message handling routines. The automatically generated SAP does not provide its own error reporting facilities, these must be placed on an error stack by routines provided by the language definer. Therefore it is necessary to insert a routine name for that purpose preceding every mandatory syntactic terminal symbol in the EBNF/WSC grammar of MODEL. SAP will pop-up the appropriate error message upon recognition that the syntactic unit reached does not match the definition in the grammar given by the EBNF/WSC.

(4) Encoding and Storing routines. These routines act as an interface between the automatically generated SAP and the information storage and retrieval subsystem (section 4.3). The invocation is done at the end of each production describing a MODEL statement type. Such routines call the store mechanism to put the collected tokens in the simulated associative memory in a coded form that facilitates later retrieval.

4.3 The Storage and Retrieval Subsystem

To facilitate the storage, retrieval and update of the non procedural statements as required by the various analysis algorithms, a self-contained and general purpose information retrieval mechanism using a simulated associative memory (AM) and a Multi-list storage organization [PRY 66] was implemented. The strings are stored "associatively" in the sense that it is possible to retrieve items based on their content. Detailed implementation description are given by [RIN 76], [CHA 77] and description of the storage entries structures used by MODEL III is found in [SHA 78].

Basically the system works as follows: during the syntax analysis each user statement is stored in the associative memory in a coded form for later retrieval and update. These operations are performed by the following routines:

(1) STORE: this routine is called upon completion of the syntax analysis of each statement. It stores in the associative memory the source language strings obtained from the user statements.

(2) RETREVE: this routine is used to access the storage entries of previously stored source statements based on some common property which is given as an expression of key words.

(3) UPDATE: this routine is used to update the key names in

an already existing storage entry previously stored in the associative memory.

Three additional routines complement the information retrieval subsystem: RETR#E, RETPNAM and RETRPRX. These are used to retrieve specific information about a particular key name in the memory.

4.3.1 The STORE Procedure

The STORE(S,D) procedure accepts strings which are formed by the subroutines called during the syntax analysis. It has two parameters, S and D. S is a string of key names to be stored and entered into a dictionary, while D is the pointer to the previously created auxiliary data which is an encoded form of the non-key source language information. The STORE procedure creates two types of entries in the memory:

- (1)-Storage entries; one is created for each MODEL statement after successful scanning of it.
- (2)-Directory entries; one is created for each key name in the string being stored.

A description of the Directory structure and key names composition is found in chapter 5 sections 5.3.1 and 5.3.2 respectively, as related to the assertion storage entry structure for cycles analysis.

4.3.2 The RETREVE procedure

The RETREVE(E,D,S,N,P) procedure search in the associative memory for storage entries satisfying the boolean expression of key names E. It also checks whether the first characters of data associated with the storage entries match the string D. The search starts at the dictionary entry S (normally the root of the directory tree) and returns a list of pointers P to those entries that satisfies the request. The total number of storage entries satisfying the logical expression of key names is returned in N.

The logical expression E used as the basis of the retrieval can be any boolean expression of key names in disjunctive normal form, where the first key name in each conjunct must be non negated.

4.3.3 The RETR#E Function

The function RETR#E(S) returns the total number of storage entries in which the key name S occur. This function is normally used in conjunction with the RETREVE procedure to obtain the size of the array of pointers N.

4.3.4 The RETRNAM Procedure

The procedure RETRNAM(S,D,N) returns in D all the key names in the directory which match with the string S, except

AD-A066 192

MOORE SCHOOL OF ELECTRICAL ENGINEERING PHILADELPHIA P--ETC F/6 9/2
USE AND EXTENSION OF AN AUTOMATIC PROGRAM GENERATOR FOR MODEL B--ETC(U)
OCT 78 J L GANA

N00014-76-C-0416

UNCLASSIFIED

78-02

NL

3 OF 6

AD
A088182



for the first two character prefix. The number of such names is returned in N.

4.3.5 The RETRPRX Procedure

The procedure RETRPRX(S,D,N) returns in D all the key names in the directory whose prefix is contained in the string S. The total number of key names found is returned in N.

4.3.6 The UPDATE Procedure

The procedure UPDATE(P,S) updates the storage entry given by the pointer P, such that the new key names given in that storage entry will be the names in the string S. The number of key names in S should be equal to the number of key names originally in the storage entry given by P.

4.4 The Precedence Matrix

The analysis of a given input specification depends very strongly on the implicit relationships defined between the statements of the module. A partial order between MODEL statements can be analyzed by representing the specification as a directed graph. The directed graph in turns can be depicted in memory as a weighted adjacency or "precedence" matrix. The matrix will contain n rows and n columns for the n nodes in the directed graph. An entry $P(I,J)$ will represent the "type" of relationship between node I and node J . Actually the precedence relationships in MODEL version III is implemented as n different "lists" instead of a matrix in order to save space and access time to its entries. For expository purposes however, these n lists can be considered to simulate the precedence matrix.

The first stage in the construction of the directed graph representation is to build a "dictionary" of data names and assertion names used in the specification. If a data name is used with different subscripts, one entry is created for each one with a different subscript expression. Assertion names can have only one dictionary entry. If a dictionary contains n entries, then the precedence matrix is generated of size $n \times n$. If the J th name is the successor of the I th name, then an entry $P(I,J) > 0$ is entered in the matrix. The actual value of the (I,J) entry represents the type of relationship holding between the two names. A

complete list of functions and procedures used to create, access and update both the dictionary of data names as well as the precedence lists can be found in [SHA 78].

Distinguishing among types of precedence relationships will play a crucial role in later stages, particularly in completeness and cycles analysis and in code generation. The different precedence relationships are explained in chapter 5 section 5.2.1 together with a summary of the precedence lists structure. An overview of the dictionary format is also presented in chapter 5 section 5.2.2 as related to the cycles analysis process.

4.5 Completeness Analysis

The functions of this component is to resolve incomplete, ambiguous or inconsistent specifications. It does so by first trying to compose, modify or delete statements automatically through analysis of the precedence matrix, and using available information in the associative memory as well as default parameters. As a second alternative it resorts to the interactive feature of the man-machine interface. It will compose error messages and will send them to the user who can then modify or add statements to the MODEL database using the Editor. The user will also be notified of the changes done automatically by the processor, so he can accept those modifications, or take them as examples which he can override with new information

through the interactive communication facility.

Among the completeness criteria required by MODEL is that of a "well formed" directed graph in the following sense* [PRY 77a]:

(1) - Rows and Columns in the precedence matrix must have the following entries:

(a) Each row and each column must have at least one element of a precedence type, with the following exceptions: columns of source media and rows of target media must not have precedence type elements. Namely, source media nodes do not have terminating directed arcs and target media nodes do not have emanating directed arcs. Also rows of field statements may have no entries (where target data is not dependent on this field).

(b) RECORD descendants of ISAM-organized FILE nodes, must have a type 7 (pointer) element in the respective column.

(c) Columns of INTERIM data names representing a variable length or a variable number of repetitions of a datum must have a type 5 or type 6 element respectively.

A failure of this criterion indicates incompleteness and a statement must be added.

(2) - The number of and types of precedence entries in rows and columns must be as follows:

(a) Each source data column or target data row can have only one type 1 or one type 2 element, respectively.

(b) An assertion row can have only one type 4 element, except where the assertion is compound.

* These completeness criteria are not exhaustive and some omissions and errors may not be recognized.

(c) An assertion column must have a number of type 3 elements equal to the number of source variables of the assertion.

(d) All precedence types must be only in the allowable elements types as given in chapter 5 section 5.2.1.

(e) Repeating data names, or their descendents, when used in assertions must have a subscript of own or closest predecessor.

Failure of this criterion indicates ambiguity and statements must be modified or deleted.

Figure 4.4 shows a refinement of the completeness analysis phase which in conjunction with the following example will be useful in illustrating some of the functions of this component.

Consider again the specification for a model of income determination where the user starts by giving the initial specification as consisting of only the following two assertions:

$$C = -6.8 + .67 * GNP;$$

$$GNP = C + I + G;$$

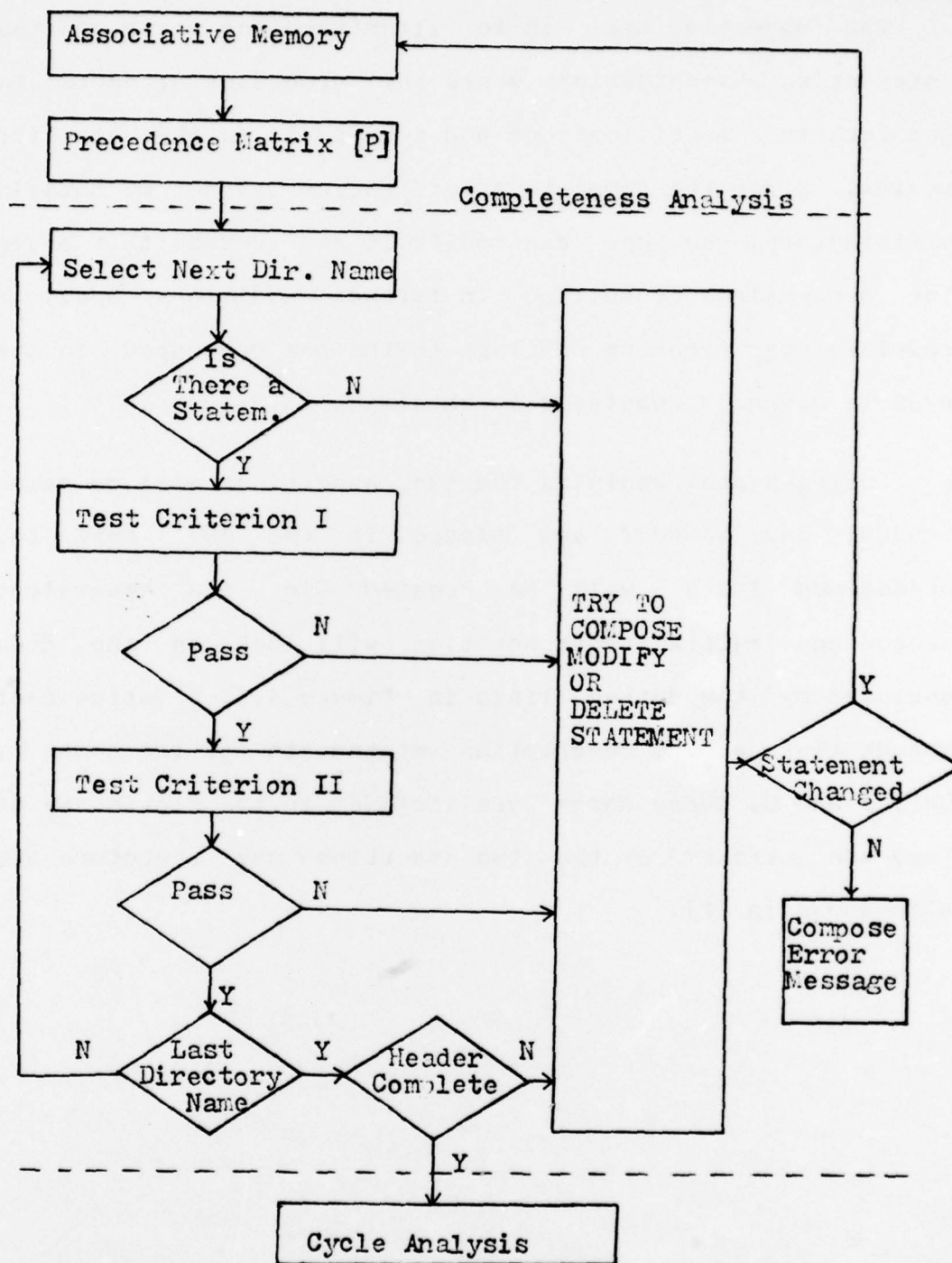


Figure 4.4

Flow diagram of Completeness Analysis Phase

The objective here is to illustrate one step of the interactive communication where the processor attempts to complete the specifications and submit it to the user for review. Since the complete specification may not be totally satisfactory, the user can modify it and resubmit it again for processing, resulting in further additions until a complete specification similar to the one presented in the example given in chapter 3 is obtained.

During syntax analysis the two assertions will be named \$AS0001 and \$AS0002 and placed in the AM. Next the precedence lists will be created and its equivalent precedence matrix representation will look as the area enclosed by the dotted lines in Figure 4.5. Notice that though there are no description statements for the names C, GNP, I and G, these names are included in the dictionary as they are referred by the two assertions and therefore are also shown in [P].

Order in Generation Statements	Order in Directory of AM	3	4	5	6	1	2	7	8	9	10	11	12
3	C						3		2				
4	G						3						
5	GNP					3			2				
6	I						3						
1	\$AS001	4											
2	\$AS002			4									
7	\$SOURCER		1		1								
8	\$TARGETR										2		
9	\$SYSIN							1					
10	\$SYSPRINT												2
11	\$DISK1									1			
12	\$DISK2												

Figure 4.5
Growing [P] for Model of Income Determination

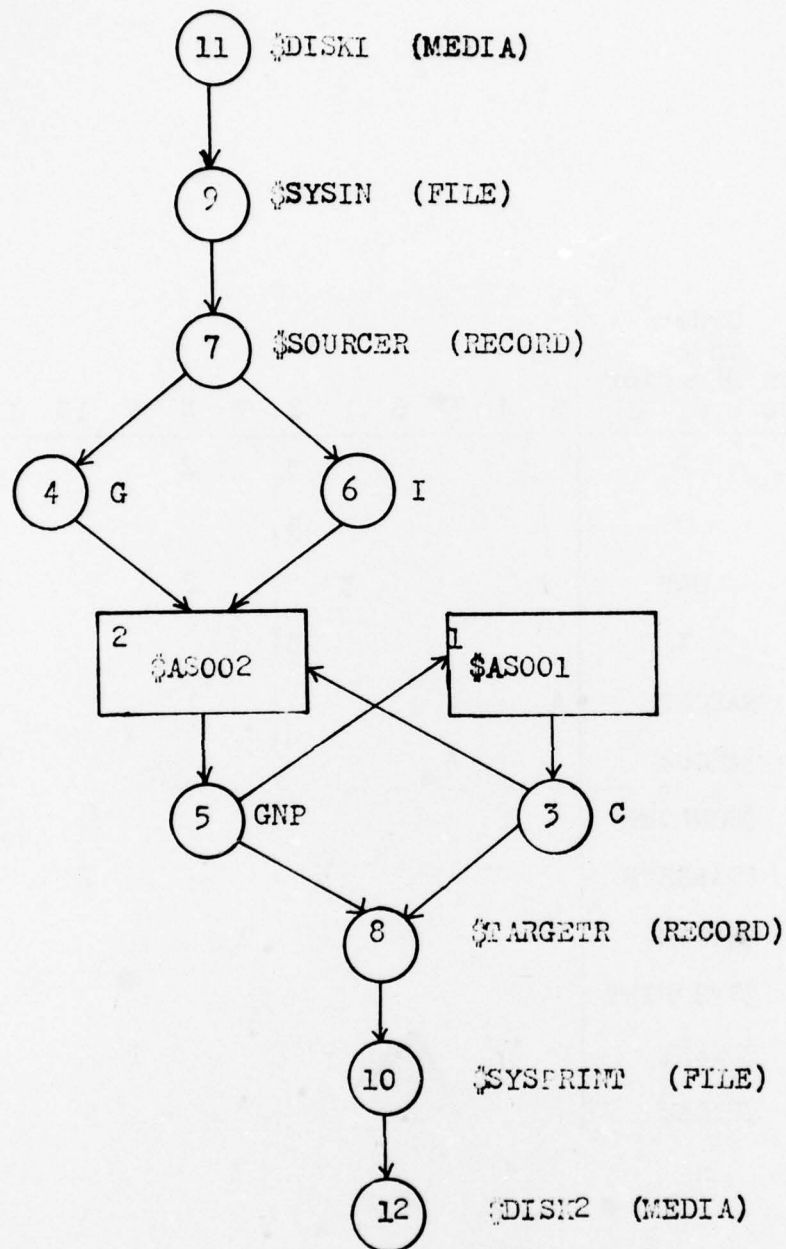


Figure 4.6

Directed Graph for Model of Income Determination
(Number indicates order of composition of statements)

COMPOSED BY	/	SAS0001: C = -6.8 + .67 * GNP;	1
USER	<	SAS0002: GNP = C + I + G;	2
	\		
	/	ON TEST	C IS INTERIM(NUM(7)); 3
	/	IS THERE	G IS FIELD(SSOURCER, NUM(7)); 4
	<	A	GNP IS INTERIM(NUM(7)); 5
COMPOSED BY	STATEMENT	I IS FIELD(SSOURCER, NUM(7));	6
PROCESSOR	?		
	\		
	/	SSOURCER IS RECORD(SSYSIN);	7
	/	ON TEST	STARGETR IS RECORD(SYSPRINT); 8
	<	OF	SSYSIN IS FILE(SDISK1, SEQ); 9
	<	CRITERIA	SSYSPRINT IS FILE(SDISK2, SEQ); 10
	1	SDISK1 IS MEDIA(DISK);	11
		SDISK2 IS MEDIA(DISK);	12
	\		

Figure 4.7

Statements for Model of Income Determination

The first step following the flow diagram in Figure 4.4 is to select the items in the AM, one at a time, according to their alphameric order in the directory of the AM. The order of composition is later shown in Figure 4.7. The first name selected is C, and since there is no descriptive statement for it, it fails the first test criterion and a statement is generated with the default parameters of INTERIM (NUM(7)), as shown in Figure 4.7. Generally data types will be selected in agreement with other data used in the rest of the specification. NUM(7) is the default in absence of any other information. Next G fails the test of whether there is a statement for it, since it is a source data name of an assertion but no target data name of any other assertion. By default it is assumed a source data FIELD. Since it has no named parent, the processor assigns a name \$SOURCER for it. Then GNP also fails the test, but since it is used as target data name, an INTERIM statement with default of NUM(7) is generated. Finally I is treated as G, that is, it is assumed a source FIELD of \$SOURCER. This process is shown in Figure 4.6 and 4.7. The directed graph in Figure 4.6 correspond to the precedence matrix [P] shown in Figure 4.5. Also the dotted area correspond to the original user specification. The nodes of the graph are numbered to indicate the order in which the statements are generated. The original specification will grow up to the complete representation as shown in Figures 4.5 and 4.6. FILE and MEDIA statements will be generated by the system

for both SSOURCER and the corresponding output record STARGETR. The final specification submitted to the user for revision is given in Figure 4.7. Though the specification may still be not totally satisfactory, it will induce the user to make further changes, additions and deletions. Note that since cycles analysis have not yet being performed, each assertion is still represented by a different node. After cycles analysis the two assertions will be linked into a single node representation indicating a simultaneous group of equations.

Figure 4.7 also shows at the left the tests from Figure 4.5 which generates the respective statements.

The criterion 1 test can also trigger generation of a new assertion. This will happen if no target data is found to be a target of an assertion. In this case the processor will search for a synonym source data, and if found, it will generate an assertion indicating equality between them.

The criterion 2 test triggers generation of:

- (1) Qualifying data names for ambiguous data names and
- (2) Subscripts of repeating data names or of the nearest predecessor of repeating data names.

Finally, after the graph is considered to be complete, the processor checks if the header consisting of module name, source and target files is present in the specification. If missing it will supply the necessary

statements as well.

The description of the completeness analysis phase is now finished, with the exception of some algorithms used in checking the consistency of subscripts in the specification. Since this task is the subject of ongoing research and is reported in [SHA 78] and [PNU 76] it will not be described here.

4.6 Cycles Analysis

This phase search for, and analyze recursive references in a specification. Recursive references between data names creates cycles in the graph representation. Because of the non-procedural nature of MODEL, these cycles are not allowed unless they legitimately represent a set of linear or non-linear simultaneous equations which potentially may be solvable by using appropriate numerical methods. Since MODEL does not explicitly allow for iteration, but instead uses subscripts in statements which are considered to be algebraic tautologies, an assertion such as: $A = A + 1$ would constitute a cycle of length one. If A were declared as both a source data name and as target data name, then the assertion would be automatically modified to: $TARGET.A = SOURCE.A + 1$, otherwise the user will have to modify his specification and "open" the cycle. More complex cycles are analyzed and either automatically "opened" by proper qualification or by soliciting such changes from the

user. Blocks of assertions representing simultaneous equations will be grouped into a common storage entry in the associative memory (representing a "single" node in the graph) , and a solution process will be generated for them at the code generation stage.

Detailed description and documentation of this phase is presented in chapter 5 of this dissertation.

4.7 Documentation

This phase automatically generates the lists, reports and tables necessary for general reference and future program maintenance. The documentation consists of three parts:

- 1) The Specification List. This is an ordered list of the given module specification divided into sections: Header, Data definition and Computation description. Each section is in turn divided into corresponding subsections: module name, source files, target files and references for the header section. Media, Files and Pointer type assertions composing the data description section. The statements in each file are indented, reflecting the tree structure of the file (refer to chapter 3 section 3.2). Interim variables, subscript parameters and assertions are all listed and every statement is numbered on the right. An example of

an specification list for the simulation example of Klein's Model I was shown in chapter 3, Figure 2.23

- 2) Cross-Reference Report. This consist of an alphanumerically ordered cross reference report of all names (data, assertions, keywords, subscripts, interim) used in the specification. Each name is listed with a brief description of its attributes and the line numbers of the statements in which it appears. Figure 4.3 shows an example of a typical cross-reference report.
- 3) Precedence List. A "precedence" or "adjacency" table is generated as a list depicting the relationships holding between a name and its successors and predecessors. A sample of the precedence list corresponding to the simulation example in chapter 3 is given in Figure 4.9

152	00000	147	CODE	IN TYPE IN TIM_SERITUT_VAN) IN CONTROL IN DISK, FIELDCHAR(4)
20	530	57	CODE	IN TYPE IN TIM_SERITUT_VAN) IN CONTROL IN DISK, FIELDCHAR(4)
4	324	20	CCCEP	IN PARAM IN DISK, GROUP
40	1000	62	COMMENT	IN REFLECTEXIST.DISK.DOCUMENT.REFREC) IN DOCUMENT IN DISK, FIELDCHAR(10)
7	330	22	CENSUPP	IN CCEFF IN PARAM IN DISK, RECORD
102	3400	114	CENSUPP	ASSERTION NAME
171	00000	161	CONTACT	IN PEINTEK, GROUP
165	00000	168	CONTROL	IN EXIST, GROUP
41	1140	7	CONTACT	IN DISK, FILE
140	00000	143	DATA	IN DISK, FIELD
30	630	49	DATA	IN DISK, FIELD
37	1430	45	DATE	IN DISK, FIELD
34	1400	23	DESCRIPTION	IN REFLECTEXIST.DISK.DOCUMENT.REFREC) IN DOCUMENT IN DISK, GROUP
159	00000	124	DISK	IN EXIST, GROUP
150	00000	334	DISK	IN EXIST, GROUP
5	230	4	DISK	IN EXIST, GROUP
169	00000	177	DOCUMENT	IN PEINTEK, GROUP
160	00000	155	DOCUMENT	IN DISK, FIELD
31	570	8	DOCUMENT	IN DISK, FILE
42	2370	72	E	IN DISK, FIELD
147	00000	142	ENDREF	IN TIM_SERITUT_VAN) IN CONTROL IN DISK, FIELDCHAR(11)

Figure 4.3

Example of Cross-Reference and Attribute Report

DICTA SUCCESSIONS		PRECEDENCE MATRIX										PRECEDENCES	
		1	2	3	4	5	6	7	8	9	10		
1	70(6)	77(6)	78(6)	79(6)	80(6)	81(6)	82(6)	83(6)	84(6)	85(6)	86(6)	30(12)	30(12)
2	64(6)	65(6)	66(6)	67(6)	68(6)	69(6)	70(6)	71(6)	72(6)	73(6)	74(6)	30(12)	30(12)
3	61(1)	62(1)	63(1)	64(1)	65(1)	66(1)	67(1)	68(1)	69(1)	70(1)	71(1)	30(12)	30(12)
4	61(1)	62(1)	63(1)	64(1)	65(1)	66(1)	67(1)	68(1)	69(1)	70(1)	71(1)	30(12)	30(12)
5	33(12)	34(12)	35(12)	36(12)	37(12)	38(12)	39(12)	40(12)	41(12)	42(12)	43(12)	30(12)	30(12)
6	14(11)	15(11)	16(11)	17(11)	18(11)	19(11)	20(11)	21(11)	22(11)	23(11)	24(11)	30(12)	30(12)
7	33(12)	34(12)	35(12)	36(12)	37(12)	38(12)	39(12)	40(12)	41(12)	42(12)	43(12)	30(12)	30(12)
8	17(11)	18(11)	19(11)	20(11)	21(11)	22(11)	23(11)	24(11)	25(11)	26(11)	27(11)	30(12)	30(12)
9	18(11)	19(11)	20(11)	21(11)	22(11)	23(11)	24(11)	25(11)	26(11)	27(11)	28(11)	30(12)	30(12)
10	23(11)	24(11)	25(11)	26(11)	27(11)	28(11)	29(11)	30(11)	31(11)	32(11)	33(11)	30(12)	30(12)
11	12(12)	13(12)	14(12)	15(12)	16(12)	17(12)	18(12)	19(12)	20(12)	21(12)	22(12)	30(12)	30(12)
12	51(1)	52(1)	53(1)	54(1)	55(1)	56(1)	57(1)	58(1)	59(1)	60(1)	61(1)	30(12)	30(12)
13	12(12)	13(12)	14(12)	15(12)	16(12)	17(12)	18(12)	19(12)	20(12)	21(12)	22(12)	30(12)	30(12)
14	36(11)	37(11)	38(11)	39(11)	40(11)	41(11)	42(11)	43(11)	44(11)	45(11)	46(11)	30(12)	30(12)
15	34(11)	35(11)	36(11)	37(11)	38(11)	39(11)	40(11)	41(11)	42(11)	43(11)	44(11)	30(12)	30(12)
16	7(12)	8(12)	9(12)	10(12)	11(12)	12(12)	13(12)	14(12)	15(12)	16(12)	17(12)	30(12)	30(12)
17	23(11)	24(11)	25(11)	26(11)	27(11)	28(11)	29(11)	30(11)	31(11)	32(11)	33(11)	30(12)	30(12)
18	24(11)	25(11)	26(11)	27(11)	28(11)	29(11)	30(11)	31(11)	32(11)	33(11)	34(11)	30(12)	30(12)
19	24(11)	25(11)	26(11)	27(11)	28(11)	29(11)	30(11)	31(11)	32(11)	33(11)	34(11)	30(12)	30(12)
20	24(11)	25(11)	26(11)	27(11)	28(11)	29(11)	30(11)	31(11)	32(11)	33(11)	34(11)	30(12)	30(12)
21	24(11)	25(11)	26(11)	27(11)	28(11)	29(11)	30(11)	31(11)	32(11)	33(11)	34(11)	30(12)	30(12)
22	24(11)	25(11)	26(11)	27(11)	28(11)	29(11)	30(11)	31(11)	32(11)	33(11)	34(11)	30(12)	30(12)
23	24(11)	25(11)	26(11)	27(11)	28(11)	29(11)	30(11)	31(11)	32(11)	33(11)	34(11)	30(12)	30(12)
24	24(11)	25(11)	26(11)	27(11)	28(11)	29(11)	30(11)	31(11)	32(11)	33(11)	34(11)	30(12)	30(12)
25	24(11)	25(11)	26(11)	27(11)	28(11)	29(11)	30(11)	31(11)	32(11)	33(11)	34(11)	30(12)	30(12)
26	24(11)	25(11)	26(11)	27(11)	28(11)	29(11)	30(11)	31(11)	32(11)	33(11)	34(11)	30(12)	30(12)
27	24(11)	25(11)	26(11)	27(11)	28(11)	29(11)	30(11)	31(11)	32(11)	33(11)	34(11)	30(12)	30(12)
28	11(12)	12(12)	13(12)	14(12)	15(12)	16(12)	17(12)	18(12)	19(12)	20(12)	21(12)	30(12)	30(12)
29	11(12)	12(12)	13(12)	14(12)	15(12)	16(12)	17(12)	18(12)	19(12)	20(12)	21(12)	30(12)	30(12)
30	11(12)	12(12)	13(12)	14(12)	15(12)	16(12)	17(12)	18(12)	19(12)	20(12)	21(12)	30(12)	30(12)
31	11(12)	12(12)	13(12)	14(12)	15(12)	16(12)	17(12)	18(12)	19(12)	20(12)	21(12)	30(12)	30(12)
32	11(12)	12(12)	13(12)	14(12)	15(12)	16(12)	17(12)	18(12)	19(12)	20(12)	21(12)	30(12)	30(12)
33	11(12)	12(12)	13(12)	14(12)	15(12)	16(12)	17(12)	18(12)	19(12)	20(12)	21(12)	30(12)	30(12)
34	11(12)	12(12)	13(12)	14(12)	15(12)	16(12)	17(12)	18(12)	19(12)	20(12)	21(12)	30(12)	30(12)
35	11(12)	12(12)	13(12)	14(12)	15(12)	16(12)	17(12)	18(12)	19(12)	20(12)	21(12)	30(12)	30(12)
36	11(12)	12(12)	13(12)	14(12)	15(12)	16(12)	17(12)	18(12)	19(12)	20(12)	21(12)	30(12)	30(12)
37	11(12)	12(12)	13(12)	14(12)	15(12)	16(12)	17(12)	18(12)	19(12)	20(12)	21(12)	30(12)	30(12)
38	11(12)	12(12)	13(12)	14(12)	15(12)	16(12)	17(12)	18(12)	19(12)	20(12)	21(12)	30(12)	30(12)
39	22(11)	23(11)	24(11)	25(11)	26(11)	27(11)	28(11)	29(11)	30(11)	31(11)	32(11)	30(12)	30(12)
40	22(11)	23(11)	24(11)	25(11)	26(11)	27(11)	28(11)	29(11)	30(11)	31(11)	32(11)	30(12)	30(12)
41	22(11)	23(11)	24(11)	25(11)	26(11)	27(11)	28(11)	29(11)	30(11)	31(11)	32(11)	30(12)	30(12)
42	22(11)	23(11)	24(11)	25(11)	26(11)	27(11)	28(11)	29(11)	30(11)	31(11)	32(11)	30(12)	30(12)
43	22(11)	23(11)	24(11)	25(11)	26(11)	27(11)	28(11)	29(11)	30(11)	31(11)	32(11)	30(12)	30(12)
44	22(11)	23(11)	24(11)	25(11)	26(11)	27(11)	28(11)	29(11)	30(11)	31(11)	32(11)	30(12)	30(12)
45	22(11)	23(11)	24(11)	25(11)	26(11)	27(11)	28(11)	29(11)	30(11)	31(11)	32(11)	30(12)	30(12)
46	22(11)	23(11)	24(11)	25(11)	26(11)	27(11)	28(11)	29(11)	30(11)	31(11)	32(11)	30(12)	30(12)
47	22(11)	23(11)	24(11)	25(11)	26(11)	27(11)	28(11)	29(11)	30(11)	31(11)	32(11)	30(12)	30(12)
48	22(11)	23(11)	24(11)	25(11)	26(11)	27(11)	28(11)	29(11)	30(11)	31(11)	32(11)	30(12)	30(12)
49	22(11)	23(11)	24(11)	25(11)	26(11)	27(11)	28(11)	29(11)	30(11)	31(11)	32(11)	30(12)	30(12)
50	22(11)	23(11)	24(11)	25(11)	26(11)	27(11)	28(11)	29(11)	30(11)	31(11)	32(11)	30(12)	30(12)

Figure 4.9

Example of Precedence List Report

4.8 Sequencing and Iteration Analysis

In this step the MODEL statements are ordered sequentially according to precedence in the directed graph representation, and the scope of iterations is determined and optimized. The ordered statements represent in fact a flowchart for the program module. The subsequent code generation component will translate this flowchart into corresponding PL/I code, thus generating the required program.

This objective is attained by means of two cooperating procedures: RANK(Pk,Sk) and ITER(Pk). RANK assigns ranks to statements and keep them in an order vector (O). ITER selects the statement names for an iteration and creates a precedence list 'Pk' for each subscript 'Sk'.

Since the graph representing the module specification is an acyclic graph after cycles analysis (with the exception of groups of simultaneous equations which are "linked" or "merged" in the AM), RANK uses a topological sorting algorithm similar to the one described by Knuth [KNU 68] to rank the statement names in the precedence list 'Pk'. After exhausting 'Pk', except for statements with other subscripts than 'k', it calls on ITER. The process starts with a call to RANK(Po,null), where Po = P, the complete precedence list of the specification and null means no subscript. ITER builds a new precedence list 'Pk+1' containing only those statement names which are candidates for iterations for

found subscripts. It then ranks a DO S_{k+1} statement and calls on RANK to sequence the statement names in ' P_{k+1} '. Upon return from RANK it ranks and END statement. The process is summarized in Figure 4.10.

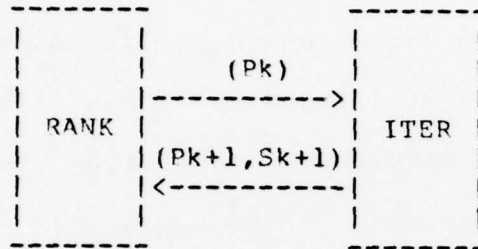


Figure 4.10
Summary of the Sequencing and Iteration Analysis Process

As will be illustrated later in chapter 5, after the sequencing stage the adjacency matrix of name relationships holding for any model specification will become block triangular. The structural causal relationships in a model will thus be obtained, with the consequent gains in efficiency for later solution procedures.

4.9 Code Generation

PL/I was chosen as the object language because of its versatility, generality of data and control structures and growing acceptability. Its language features make it suitable for general business data processing as well as for model building applications in engineering and social sciences.

The code is generated directly from the flowchart produced in the sequencing and iteration analysis. In general it is a straight forward translation task using known methodology (see for example [RIN 76] for a detailed description). Data description statements are used to generate corresponding PL/I declarations. Next single assertions are translated into procedure calls. Assertions belonging to a set of simultaneous equations are grouped into a solution procedure to be described in detail in chapter 6 of this dissertation. Finally the flowchart entries are taken one by one and translated into appropriate PL/I code. A sample summary to illustrate this translating process is given below:

- 1) Media and file statements are translated into JCL statements and into open and close file code.
- 2) Source record statements are translated into READ record code.
- 3) Target record statements are translated into WRITE

record code.

- 4) FOR <subscript> statements are translated into DO <subscript> 1 to N, where N is the number of repetitions.
- 5) END statements remain unchanged.
- 6) POINTER.SOURCE interim variables are translated into search procedures calls.

etc.

4.10 Compilation and Execution of the Generated Program

The PL/1 program generated by the MODEL processor is next submitted to the PL/1 Optimizer Compiler for translation into an efficient host object language for final execution. The optimizer compiler will perform low level optimization such as: elimination of common arithmetic expressions, transfer of constant expressions from the scope of loops, reduction of logical expressions, etc. (refer to the PL/1 reference manual [PL1 75] for detailed account of the different levels of optimization).

The program design, code generation and optimization done by the MODEL processor, in addition to the program and object code optimization of the PL/1 compiler should produce an efficient and reliable program ready for execution. This will include a numerical solution procedure for linear or

non-linear simultaneous equations normally found in dynamic models. The documentation produced automatically by MODEL, together with its interactive communication facility should substantially reduce debugging time for every stage of model development.

CHAPTER 5

Cycles Analysis

5.1 Introduction

This chapter describes the methodology and algorithms used in the cycles analysis phase of the MODEL processor. The set of MODEL statements used to describe a module are translated into a "directed graph" representation. The nodes of this graph are either data names or assertions. The directed arcs connecting these nodes define a precedence relationship between them, which may be of different types, depending upon the nature of the successor and predecessor nodes involved. This directed graph representation can be pictured as a precedence matrix P (also called a weighted adjacency matrix). An entry $P(i,j)$ in this matrix will contain the precedence type between node i (predecessor node) and node j (successor node). The cycles processor analyses the precedence matrix representation of the module graph in order to detect circular definitions. Since MODEL is a non procedural language which do not use "control" statements or iterative DO loops, these circularities or cycles will generally not be allowed, with the exception of those occurring between assertions and their data names which are treated as a simultaneous equation system and stored as a single recursive group.

Cycles between elements with precedence types other than

those relating variables and assertions are reported as errors. The user should modify his statements in order to delete the corresponding pointers, opening in this manner the loops. Also since assignment statements are not permitted in the language, some trivial loops between variables of an assertion such as $A = A + 1$ (a loop of length one) are reported for correction. In modeling and other scientific applications where it is intended to describe systems with interactions between the component subsystems (i.e., feedback systems), it is perfectly legitimate to define non-procedurally linear or non linear sets of assertions representing such systems and which form a group of simultaneous equations that can be solved by use of standard techniques such as Cramer's rule or elimination method if they are linear, or by means of iterative procedures such as Gauss-Seidel or Newton's if they are non linear. The assertions that describe these interactive systems will form compound cycles in the graph representation and will be detected and reported to the user as part of the cycles analysis.

Before turning into the description of the algorithm implementation, some basic notions on causal systems as well as the necessary definitions and graph theoretical nomenclature used in the rest of the chapter, are introduced in the next two sections.

5.1.1 Causal Ordering

In analysing complex model structures, it is possible to find subsets of the total system under study which can determine the value of endogenous variables independently from the rest of the system or model. Once those subcomponents are identified, it is possible to solve and study them in isolation from the rest of the model with the consequent reduction in complexity. The hierarchical relationship holding between the subcomponents is also important since it helps to understand the working of the system. For instance it is desirable to verify if there exists a one-way causal relationship in which the variables in one subsystem affects the behavior of the total model but not vice versa. Simon [SIM 53] gives a formal treatment of this subject, with applications to economics and in particular to the problem of identifiability. Later Ando and Fisher [AND 63] in studying the structure of dynamic systems in the social sciences stated:

"In RECENT YEARS economists have come to deal with dynamic systems of increasing size and complexity. The difficulty of analyzing such systems, however, appears to increase faster than does our ability to handle them with the aid of high speed computing equipment. As a consequence, it is becoming more and more important to secure information on the nature of those aspects of a system which, when present, enable us to treat a part of it separately from the rest or to deal with the relationships among particular subsystems as though it were independent of the structures within those subsystems."

The same authors provided some theorems which served as guides for the analysis and interpretation of dynamic systems on which a partition or a semi-decomposition can be applied, and resolve some questions on the problem of aggregation of variables in estimating and simulating these interrelated systems. This section will provide a framework for the methodology employed by MODEL which follows.

Since the relationships between variables in MODEL is represented by means of a precedence matrix, a binary matrix 'A' can be generated from this with entries

$$A(i,j) = 1 \text{ if and only if } P(i,j) > 0 \\ \text{otherwise } A(i,j) = 0$$

Using this representation it is possible to analyze the structure of a system by noticing that any two elements in it (ex. endogenous variables) $s(i)$ and $s(j)$ are related in a certain way, or they are not. That is

$$\text{either } s(i)Ps(j) \text{ or } s(i)\bar{P}s(j)$$

5.1.1.1 Definitions

An n equation system in n independent and consistent endogenous variables is said to be self-contained or determinate. A self-contained system of equations can be further categorized as:

1) Decomposable, if it can be partitioned into subsystems which in turn are also self-contained.

2) Causal, if it is not decomposable but contain at least one self-contained subsystem.

3) Indecomposable, if it has no self-contained subsystems

By suitable rearrangement of the order of equations and variables in the system, the matrix of a decomposable system can be made block-diagonal, while that of a causal system can be made block-triangular.

In general the above definitions can be extended to any system whose interrelations are represented by a binary matrix, including those systems composed of several sets

$$S = S_1 \cup S_2 \cup \dots \cup S_n$$

which have been compounded into a single one. One of the objectives of this analysis is precisely to study the feasibility of partitioning the totality into smaller and more manageable constituents.

Example 1:

	(1)	(2)	(3)
(1)	1	0	0
(2)	0	1	1
(3)	0	1	1

The system represented by the above binary matrix can be partitioned into two self-contained subsystems. One containing element {(1)} alone, and the other consisting of the set {(2),(3)}. The entire system is decomposable

Example 2:

	(1)	(2)	(3)	(4)
(1)	1	1	0	0
(2)	1	1	0	0
(3)	1	0	1	1
(4)	0	1	1	1

This system is a causal one, since even though elements $\{(1), (2)\}$ constitute a self-contained subsystem, they in turn affect the subsystem composed by elements $\{(3), (4)\}$, establishing in this way a causal relationship.

Example 3:

	(1)	(2)	(3)	(4)
(1)	1	0	1	0
(2)	0	1	1	1
(3)	1	1	0	1
(4)	0	0	1	1

The above structure does not contain any self-contained subsystem, and is therefore indecomposable. All the elements interact with each other. If the elements of such system were endogenous variables of a model, then the solution to it can be found by simultaneously solving all the equations of the system. If the system were a program consisting of references between functions, procedures and data names in the vocabulary of the programming language, then the matrix will represent a recursive definition

between the elements which indicates either iteration or the need of stacks or push-down processors in order to implement or 'solve' the program. At this level of abstraction these concepts are equivalent.

In a causal system it is possible to introduce a precedence relationship which in turn determines the recursive structure of the system under study, as far as solution is concerned.

When working in a system which allows for incrementality, for instance when integrating several independently built models, the recursive structure can become tremendously complex and only by means of an automatic mechanism it is possible to get the necessary insight into its nature.

5.1.1.2 Block Diagonalization and Block Triangularization of Matrices

A self-contained subsystem which does not contain any other self-contained subsystem is called a minimal self-contained subsystem or a complete subsystem of zero order.

It is then possible to define recursively a complete subsystem of n th order as a self-contained subsystem which becomes minimal after eliminating the elements found in the complete subsystem of $n-1$ order. For purposes of stating the algorithm let

\mathcal{S}^n be the class of complete subsystems of n th order.

$S = \{s_1, s_2, \dots, s_m\}$ the system under study.

$R(s)$ the reachability set of s , consisting of all elements of S lying on paths that originate from s .

$A(s)$ the antecedent set of s , consisting of all elements of S which include s , but which do not originate from s .

then

$$R(s) \in \mathcal{S}^0 \iff R(s) \cap A(s) = R(s) \quad \forall s \in S$$

From this definition it is clear that at each hierarchical level (order) there may be more than one complete subsystem, but if that is the case they are disconnected. That is, any two elements (s_i) and (s_j) in S which belong to the same level are either not connected or there is a two way connection (strongly connected elements) between them. If the latter is the case, they must belong to the same complete subsystem.

Once the complete subsystems of zero-order are found, it is possible to eliminate all their elements from the binary matrix representation

$$S' = S - \{s \mid s \in \mathcal{S}^0\} \quad \forall s \in S$$

and then it is possible to look for minimal self-contained subsystems in the reduced matrix S' . This will identify those complete subsystems of first order. The process is

carried out until the reduced matrix becomes empty. The matrix can also be rearranged to show the separate blocks of the partition as well as the hierarchical structure of the system.

ALGORITHM:

1. Start by looking for complete subsystems of zero order. If there are none, the system is indecomposable.
2. Renumber sequentially the columns as well as the rows of non-zero elements which are minimal self-contained subsystems.
3. Eliminate those rows and columns from the matrix representation and go back to the first step unless no reduced matrix is left.

The following example will illustrate the development of a block triangular matrix given the adjacency matrix in Table-5.1

	1	2	3	4	5	6
1	1	0	1	0	0	0
2	0	1	0	0	0	0
3	1	0	1	0	0	0
4	0	1	0	1	0	1
5	0	1	0	0	1	0
6	1	0	0	1	1	1

Table-5.1
Matrix for example

In order to obtain the reachability set $R(s)$, as well as the antecedent set $A(s)$, it is more convenient to generate from the bit adjacency matrix a "path" or "reachability" matrix R . The reachability entry is 1 if there is some path of any length from i to j

$$R(i,j)=1 \iff \exists k_1, k_2, k_3, \dots, k_n \text{ such that}$$

$$A(i, k_1) = A(k_1, k_2) = \dots = A(k_n, j) = 1$$

$$R(i,j)=0 \text{ otherwise}$$

This could also be defined as

$$A \vee A^2 \vee A^3 \vee \dots \vee A^n = R,$$

the transitive closure of A , where $A^k = A \wedge^{k-1} A$ and the boolean matrix operations $X = A \wedge B$ and $Y = A \vee B$ for square matrices of order n are defined as:

$$X(i,j) = \bigvee_{k=1}^n (A(i,k) \wedge B(k,j))$$

$$Y(i,j) = A(i,i) \vee B(i,j)$$

Warshall's algorithm [WAR 62] can be used to generate the reachability matrix in $O(n^3)$ as follows:

ALGORITHM: 1.- $R = A$

2.- $\forall i, j$

$$R(i, j) = 1 \implies R(i, k) = R(i, k) \vee R(i, j) \wedge R(j, k)$$

The reachability matrix corresponding to the example is given in Table-5.2 below

	1	2	3	4	5	6
1	1	0	1	0	0	0
2	0	1	0	0	0	0
3	1	0	1	0	0	0
4	1	1	1	1	1	1
5	0	1	0	0	1	0
6	1	1	1	1	1	1

Table-5.2
Reachability Matrix for Example

The presence of cycles can be detected by examining the diagonal elements of the reachability matrix R . Node i will lie on a cycle if and only if $R(i, i) = 1$. In Table-5.2 it is seen that every element in the diagonal is 1, hence every node of the corresponding graph contains a self loop.

Element $\{2\}$ is zero order since

$$R(2) = R(2)A(2) = \{2\}\{2, 4, 5, 6\} = \{2\}$$

The set consisting of elements $\{1,3\}$ is also zero order since

$$R(1)=R(3)=R(1)A(1)=\{1,3\}\{1,3,4,6\}=\{1,3\}$$

If the system were representing a model of six equations in six endogenous variables, then equations 1 and 3 could be solved independently. Those columns and rows get renumbered and eliminated from the representation for the next pass of the process as depicted in Table-5.3

	2	1	3					ORDER
	1	2	3	4	5	6		
2	1	0	1	0	0	0		0
1	0	1	0	0	0	0		0
3	1	0	1	0	0	0		0
4	1	1	1	1	1	1		
5	0	1	0	0	1	0		
6	1	1	1	1	1	1		

Table-5.3
Identification of Zero-order Elements

Table-5.4 shows that element $\{5\}$ is first order and finally Table-5.5 identifies the set $\{4,6\}$ as being of second order. Again the respective columns and rows get renumbered and their entries eliminated from the representation.

	1	2	3	4	5	6	ORDER
1	1	0	1	0	0	0	0
2	0	1	0	0	0	0	0
3	1	0	1	0	0	0	0
4	1	1	1	1	1	1	
5	0	1	0	0	1	0	1
6	1	1	1	1	1	1	

4

Table-5.4
Identification of First-order Elements

	1	2	3	4	5	6	ORDER
1	1	0	1	0	0	0	0
2	0	1	0	0	0	0	0
3	1	0	1	0	0	0	0
4	1	1	1	1	1	1	2
5	0	1	0	0	1	0	1
6	1	1	1	1	1	1	2

Table-5.5
Identification of Second-order Elements

Sorting the new numbers for rows and columns gives the final block diagonal representation shown in Table-5.6 for the original binary adjacency matrix.

	2	1	3	5	4	6
2	1	0	0	0	0	0
1	0	1	1	0	0	0
3	0	1	1	0	0	0
5	1	0	0	1	0	0
4	1	0	0	0	1	1
6	0	1	0	1	1	1

Table-5.6
Final Ordering

The same information can be depicted as an acyclic directed graph, as shown in Figure 5.1. In order to draw this graph it is necessary to collect all elements which are interconnected at the same class level into a single vertex while the relationships between nodes at different levels is represented by directed arcs.

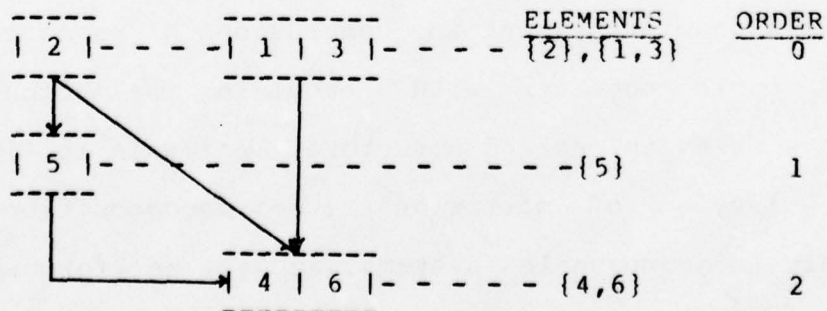


Figure 5.1
Digraph of the Matrix in Table-5.6

In solving a model, the order or hierarchy will give the appropriate sequence of execution. Those subsystems that belong to the same order class can be solved in parallel. Also savings in computation time can be realized if the scope of the iterative solution procedures is restricted to those nodes containing a set of simultaneous equations in the same class. No attempt is made at present by MODEL to guarantee the existence of a solution before generating a solution process for a block or constituent of a partition. From an analytical point of view MODEL restricts itself to just structural analysis of the given model, based on binary relations between the elements of the model. Since it is recognized that a partition and ordering based solely in structural analysis, without considering the "weight" of every element (variable) involved in the relationship in individual detail, might not be totally appropriate as far as solution is concerned for some cases, a mechanism is provided which allow the user to override the cycles analysis grouping for the generation of a solution process (refer to chapter 3 section 3.4.2). Also section 5.4 at the end of this chapter present the conclusions of research done in this topic together with providing an account of algorithms used to extend structural analysis to include further levels of partition (semi-decomposition) in completely indecomposable systems, as well as for ordering inside a simultaneous system. However for purposes of management of an information system as well as to allow the

automatic generation of programs with properties of sharing and incrementality, MODEL provides the necessary framework and basic mechanisms in which other algorithms based on symbolic algebraic analysis (variable substitution) [SOY 71] as well as numerical analysis (ex. sensitivity analysis) could be added to extend the application universe as well as for completeness.

While the algorithm proposed in this section illustrate the concepts involved in both identifying cycles as well as groups of simultaneous assertions, it is not an efficient algorithm nor does it takes into consideration the particular representation of relationships between all the elements in MODEL language. For instance, the precedence matrix in MODEL is given as a list structure in order to save both space and search time. Also the cycles analysis will only identify groups with "two-way" relationships and will group them together to form an acyclical directed graph representation. The hierarchical ordering is done in a later stage by the "sequencing" analysis phase, which besides of a topological ordering of the statements for proper execution, performs an iteration analysis by grouping and determining the "scopes" of all elements using subscripts which need to be iterated.

5.1.2 Graph Theory Background

A directed graph $G=(V,E)$, consisting of a set of vertices V and a set of edges E , is said to be STRONGLY CONNECTED if there is at least one directed path from every vertex to every other vertex. A path $p:v \xrightarrow{*} w$ in G is a sequence of vertices and edges leading from node v to node w (v and w in V).

$$S_C(G) \iff \{\exists p_1 \exists p_2 \mid p_1 : v \xrightarrow{*} w \ \& \ p_2 : w \xrightarrow{*} v, \ (v,w) \in V\}$$

A digraph G is said to be WEAKLY CONNECTED if its corresponding undirected graph is connected but G is not strongly connected.

An equivalence relation R can be defined on the set of vertices as follows:

$$vRw \iff \{\exists p \mid p : v \xrightarrow{*} w \text{ and } w \in p\}$$

Let the distinct equivalence classes under this relation be V_i ($1 \leq i \leq n$) and let $G_i=(V_i, E_i)$ where $E_i=\{(v,w) \in E \mid v,w \in V_i\}$ then it can be shown that

- i) Each G_i is strongly connected and
- ii) No G_i is a proper subgraph of a strongly connected subgraph of G .

Each maximal connected (weakly or strongly) subgraph of a digraph G is called a COMPONENT of G . Within each component of G the maximal strongly connected subgraphs G_i are called the strongly connected COMPONENTS or FRAGMENTS of G .

For example in Figure 5.2 the graph shows the fragment $F = \{3, 4, 5, 6\}$

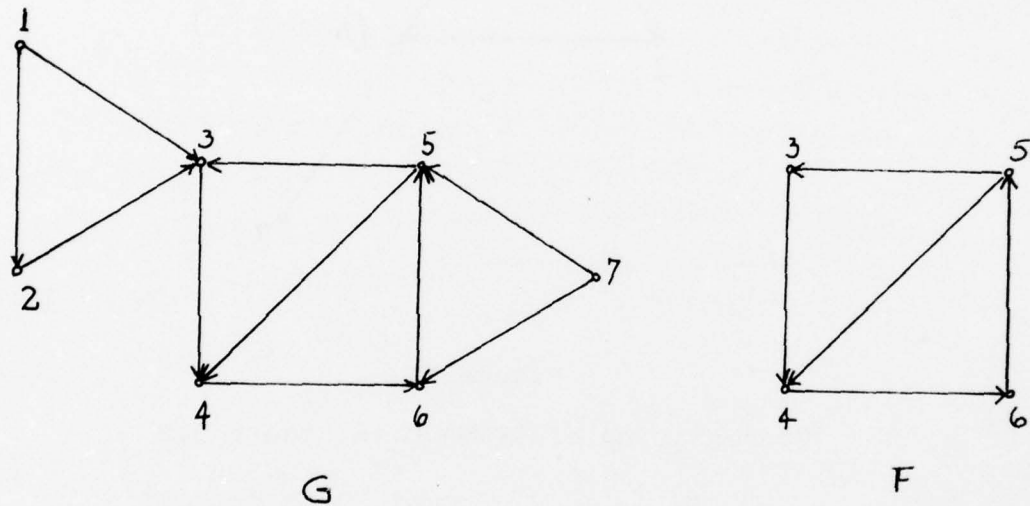


Figure 5.2
Graph G with Fragment F

The CONDENSATION G_c of a digraph G is a digraph in which each strongly connected fragment is replaced by a vertex, and all directed edges from or to such fragment from other

elements of the component are replaced by a single directed edge.

Figure 5.3 shows the condensation of the graph in Figure 5.2

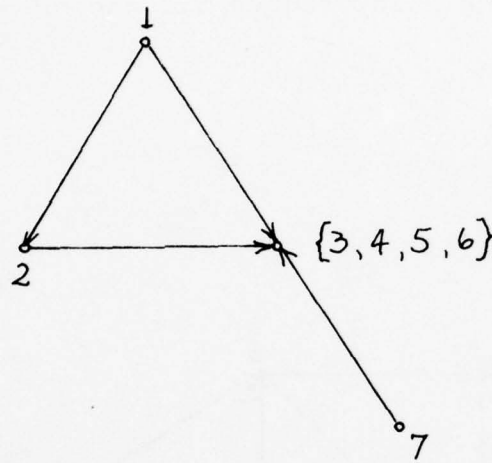


Figure 5.3

Condensation of Graph G in Figure 5.2

Clearly the condensation of a digraph has no direct circuits.

The problem of finding the strongly connected components of a digraph is related to the problem of identification of "compound cycles" in the MODEL processor [PRY 77a], since these compound cycles may represent proper simultaneous assertions for which a solution can be found using appropriate numerical methods.

There are some instances in which the enumeration of

every single circuit or cycle could prove of some advantage. For example in section 5.4 of this chapter some algorithms are proposed in order to "tear" or "semi-decompose" large systems of equations based on occurrences of individual cycles inside a strongly connected fragment. This type of procedure could be used for more efficient handling and management of complex systems.

Several algorithms are given in the literature to find all elementary circuits of a digraph [DEO 76]. In particular Rin [RIN 76] uses one by Bersztiss [BER 71] in his implementation of MODEL. A more efficient algorithm which can be readily adapted to the present implementation using a list structure representation for the adjacency matrix is found in Johnson [JOH 75]. However, in general, the enumeration of all elementary circuits can become very time consuming and will not normally provide much interesting information. For this reason the search for strongly connected fragments is preferred as the standard cycle analysis algorithm. Further, if in future implementations an algorithm for elementary cycles is needed, it will be more efficient to start the search from the already identified individual fragments.

5.1.3 Algorithms for Graph Manipulation

Four classes of algorithms for the enumeration of all circuits of a graph are generally described in the literature [DEO 76]. These are:

- i) circuit vector space for undirected graphs;
- ii) search algorithms;
- iii) powers of adjacency matrix; and
- iv) edge-digraph.

For the identification of strongly connected components of a graph however, only algorithms in classes ii) and iii) are considered, both because of efficiency reasons as well as because the ease of implementation in the existing structure of the MODEL processor.

A description of an algorithm using powers of the adjacency matrix was already given in section 5.1.1.2. The first step consists in using Warshall's algorithm to generate the path or reachability matrix from the given adjacency structure. Warfield [WAR 73,74] gives an excellent survey on the use of binary matrices in system modeling and describes the algorithm in which every element in the matrix is associated with two sets: a "reachability" set and the "antecedent" set. It is then shown that the intersection (set product) of these two sets identifies the elements of the partition or equivalent class. A similar algorithm is given in [GAN 76a] which takes advantage of a "two-tailed" reduction procedure applied recursively to the

path matrix before identifying the strongly connected elements at each level using the properties of "reflexivity", "symmetry" and "transitivity" of equivalence relations.

No matter how good a matrix representation of a graph can be for either formal analytical or expository purposes, its main drawback lies in the order of complexity of all algorithms associated with this representation. The time and space bounds associated with this algorithms is at least $O(n^2)$, and for our application will go up to $O(n^3)$ or $O(n^4)$ at the best. Many authors have found algorithms which are proportional to the number of vertices and edges of the graph. The core of these algorithms is to use a list structured representation of the graph* and some type of search technique in an appropriate search space.

Backtracking or depth-first-search (DFS) is a widely known technique used in problem solving and artificial intelligence [NIL 71]. The process begins by choosing any starting vertex of G and traverse it by always selecting an edge emanating from the vertex most recently selected which still have not used edges in its adjacency list.

* The structural representation of a graph $G=(V,E)$ given as an adjacency list of vertices requires an adjacency list $A_g(v)$ for each $v \in V$.

Then

$$u \in A_g(v) \iff (v,u) \in E$$

The process is easy to program recursively if a stack is used to keep track of the vertices with possible not-used edges emanating from it.

Next section describes the actual implementation of a recursive Depth-First-Search algorithm to find all fragments of the digraph structure representing MODEL statements.

5.2 Identification of Strongly Connected Components in MODEL

5.2.1 Structure of the Precedence List

The precedence matrix in model is of size $n \times n$, where n is the number of dictionary entries used in the MODEL specifications. Any element $P(i,j) > 0$ in the precedence matrix indicates a relationship between predecessor dictionary entry i and successor dictionary entry j which can be put into one of the following three classes [SHA 78]:

- i) Hierarchical relationship - a relation between a data name and its descendants (or its parents).
- ii) Value dependency relationship - a relation between a data name and an assertion name.
- iii) Pointing relationship - a relation between an EXIST, LENGTH or POINTER name and a data name.

Strongly connected components representing assertions to be solved simultaneously can occur only between elements

with precedence types in class ii) above. The description of each precedence type is given below:

- Type 1: Indicates a hierarchical relationship between the source data names and its descendants.
- Type 2: Representing also a hierarchical relationship; between the target data names and their parent nodes.
- Type 3: A value dependency relationship between the source variables of an assertion and the assertion itself.
- Type 4: Also a value dependency relationship; between an assertion and target variables used in that assertion.
- Type 5: Indicating a pointing relationship between a "length name", L, and a data name whose length is given by L.
- Type 6: Also a pointing relationship; between an "exist name", E, and a data name whose size depends on E.
- Type 7: A pointing relationship between a "pointer name", P, and a repeating data name whose instance is determined by P.
- Type 8: A hierarchical relationship between entries for different Subscripted variable representations.

Figure 5.4 below gives as example the graphic representation for the assertions:

ASN1: $C = - 6.8 + .67 * GNP ;$

ASN2: $GNP = C + I + G ;$

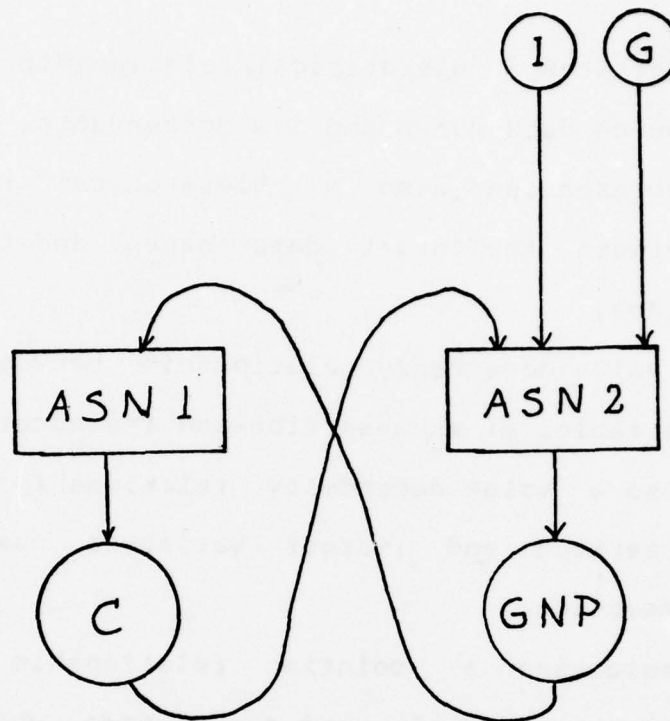


Figure 5.4
Directed Graph Representation
for Two Assertions in MODEL

The assertion nodes are depicted by square boxes while the data names nodes are represented by circles. Notice that a cycle (strongly connected or maximal) is defined by this two assertions since the target (endogenous) variable of one assertion is used as source variable for the other and vice-versa, a characteristic of every simultaneous equation system. The equivalent precedence matrix representation is given in Table-5.7

	C	GNP	I	G	ASN1	ASN2
C	0	0	0	0	0	3
GNP	0	0	0	0	3	0
I	0	0	0	0	0	3
G	0	0	0	0	0	3
ASN1	4	0	0	0	0	0
ASN2	0	4	0	0	0	0

Table-5.7

Precedence Matrix for Graph in Figure 5.4

Because of the complexity of searching algorithms that use a matrix representation, and also since the precedence matrix is a sparse one, to conserve space the matrix is not kept as an array, but rather as n lists, corresponding to the n dictionary entries. The i th precedence list contain all non-zero entries for the i th row and i th column of the equivalent precedence matrix. The lists can be described by the following PL/1 structure:

```

DCL 1 ADJM ENTRY BASED (ADJMP),
    2 ALLOC BIN FIXED,
    2 RUSED BIN FIXED,
    2 CUSED BIN FIXED,
    2 ENTRIES (N REFER (ALLOC)),
    3 COL BIN FIXED,
    3 CVALUE BIN FIXED,
    3 ROW BIN FIXED,
    3 RVALUE BIN FIXED;
```

which correspond to the following pictorial structure:

```
-----
|ALLOC|RUSED|CUSED||COL(1)|CVALUE(1)|ROW(1)|RVALUE(1)|- -
-----
```

```
- - -||COL(ALLOC)|CVALUE(ALLOC)|ROW(ALLOC)|RVALUE(ALLOC)||
-----
```

where

ALLOC gives the size of the list. It is equal to the maximum number of columns or rows with non-zero entries for the particular list.

$$\text{ALLOC} \geq \text{MAX}(\text{CUSED}, \text{RUSED})$$

RUSED is the number of non-zero entries in the row for the dictionary element.

CUSED gives the number of non-zero entries in the column for the dictionary entry.

COL(I) (I=1 to RUSED) gives the column number for the non-zero entries under the dictionary name row.

CVALUE(I) (I=1 TO RUSED) is the actual precedence type corresponding to COL(I) for the dictionary element under consideration.

ROW(I) and RVALUE(I) (I=1 TO CUSED) similarly gives the row numbers and contents corresponding to the non-zero matrix entries for the given dictionary element column.

For instance the list corresponding to the dictionary entry ASN2 with precedence matrix entries $P(1,6)=3$, $P(3,6)=3$, $P(4,6)=3$ and $P(6,2)=4$, will look as follows:

```
-----
|3|1|3||2|4|1|3|-|-|3|3|-|-|4|3||
-----
```

The function ACRDRA(I) returns a pointer to the precedence matrix list, ADJMP, for the Ith dictionary entry.

5.2.2 Dictionary for the Precedence Lists

In order to access the lists that contains the "precedence matrix entries" for each data name or assertion name supplied by the user, a dictionary is created with the following format:

```
|-----+-----+-----+-----+-----|
|STEPTR|SUB-PTR|SUB#|NEXT-INDEX|ADJMPTR|
|-----+-----+-----+-----+-----|
```

where

STEPTR is the storage entry pointer of the data name represented by this dictionary entry;

SUB-PTR is a pointer to the subscript list, associated with that data name;

SUB# is the dimensionality of the subscript given by SUB-PTR;

NEXT-INDEX is the next dictionary index of the same data name, but with different dimensionality and subscript;
finally

ADJMPTR is the pointer to the precedence list for the dictionary entry.

A data name can have more than one dictionary entry in the following cases:

(1) The data name is a descendant of an update file;
or

(2) The data name is an array. In this latter case, each use of the data name with different subscript has an entry in the dictionary.

An assertion name has only one entry in the dictionary.

5.2.3 Algorithm STRGCON

The following is an adaptation of a well known an efficient linear DFS algorithm by Tarjan [TAR 72]:

Whenever a DFS is performed on a graph $G(V,E)$, the edges generated while traversing fall into four classes:

i) Those which lead to a new vertex and hence form a tree.

ii) Those running from ancestors to descendants in the tree. These can be ignored since they do not affect the strongly connected components of G .

iii) Those running from descendants to ancestors in the tree. These are called FRONDS.

iv) Finally those from one subtree to another in the tree. These are called CROSS-LINKS.

If the vertices of G are numbered according to the traversing order, then

$\text{CROSS-LINK}(v,w) \implies \text{NUMBER}(v) > \text{NUMBER}(w)$

A DFS will create a set of trees containing all the vertices of G , called the "spanning forest" of G , and sets of fronds or cross-links (other edges are not needed). A directed graph consisting of a spanning forest and sets of fronds and cross-links is called a JUNGLE.

If the vertices are numbered in the order they are reached during the search, and are referred by their number, then we have the following LEMMA.

Let $\{v,w\} \in V$ and also $p_1: v \xrightarrow{*} w$ and $p_2: w \xrightarrow{*} v$ (that is v & w belong to the same strongly connected component).

Let F be a spanning forest generated by repeated DFS of G . Then v & w have a common ancestor in F . Further if u is the highest numbered common ancestor of v & w , then

$\{\exists p_1 \exists p_2 \mid p_1: u \xrightarrow{*} v \text{ \& } p_2: v \xrightarrow{*} u\}$

that is u is in the same strongly connected component as v and w .

Proof: without loss of generality assume $v \leq w$. Let $p: v \xrightarrow{*} w$ be a path in G . Let 'Tu' with root 'u' be the smallest subtree of a tree in F containing all the vertices in p . Such tree exists, since p can pass from one tree in F to another tree with smaller numbered vertices but p can never lead to a tree with larger number vertices. If p were contained in two or more trees of F , it could not end at w , since $v \leq w$. Thus 'Tu' exists and v and w have a common ancestor in F . The rest of the lemma is immediate if either $u=v$ or $u=w$. Otherwise let 'Tu1' and 'Tu2' be two distinct subtrees containing points of p such that

$u \xrightarrow{*} u_1$ and
 $u \xrightarrow{*} u_2$

If only one such subtree exists then u is since 'Tu' is minimal. Otherwise p can get from u_1 to u_2 only passing through vertex u , since u is the highest numbered common ancestor to both. Hence the lemma holds.

Corollary: Let C be a strongly connected component in G . Then the vertices of C

define a subtree of a tree in F , the spanning forest of G . The root of the subtree is called the root of the strongly connected component C .

The problem of finding the strongly connected components of a graph G thus reduces to finding the roots of the strongly connected components.

The following procedure will determine if a vertex is the root of a strongly connected component. Let again $v \Rightarrow^* w$ denote a tree edge and let $v \dashrightarrow$ denote a frond or cross-link, then

$$\text{LOWLINK}(v) = \min(\{v\} \cup \{w \mid v \dashrightarrow^* w \ \& \ \exists u (u \Rightarrow^* v \ \& \ u \Rightarrow^* w \ \& \ u \text{ and } w \text{ are in the same strong connected component of } G)\})$$

That is $\text{LOWLINK}(v)$ is the smallest vertex which is in the same component as v and is reachable by traversing zero or more tree arcs followed by at most one frond or cross-link.

then

$$v \in \text{ROOT}(C) \iff \text{LOWLINK}(v) = v$$

Proof: if v is the root of a strongly connected component then $\text{LOWLINK}(v) = v$ by definition. Conversely assume u is the root of C , and v is a vertex different than u , there must be a path $v \Rightarrow^* u$, consider the first edge of this path which leads to a vertex w not in T_v , a subtree, this must be either a frond or cross-link and therefore $\text{LOWLINK}(v) \leq w < v$, since the highest numbered common ancestor of v and w is in C .

Tarjan's algorithm is $O(V, E)$ when given the graph structure as an adjacency list. The flowchart in Figure 5.5 illustrate the basic stages of the algorithm. Appendix B presents the complete PL/I source listing of the Cycles processor as implemented in MODEL.

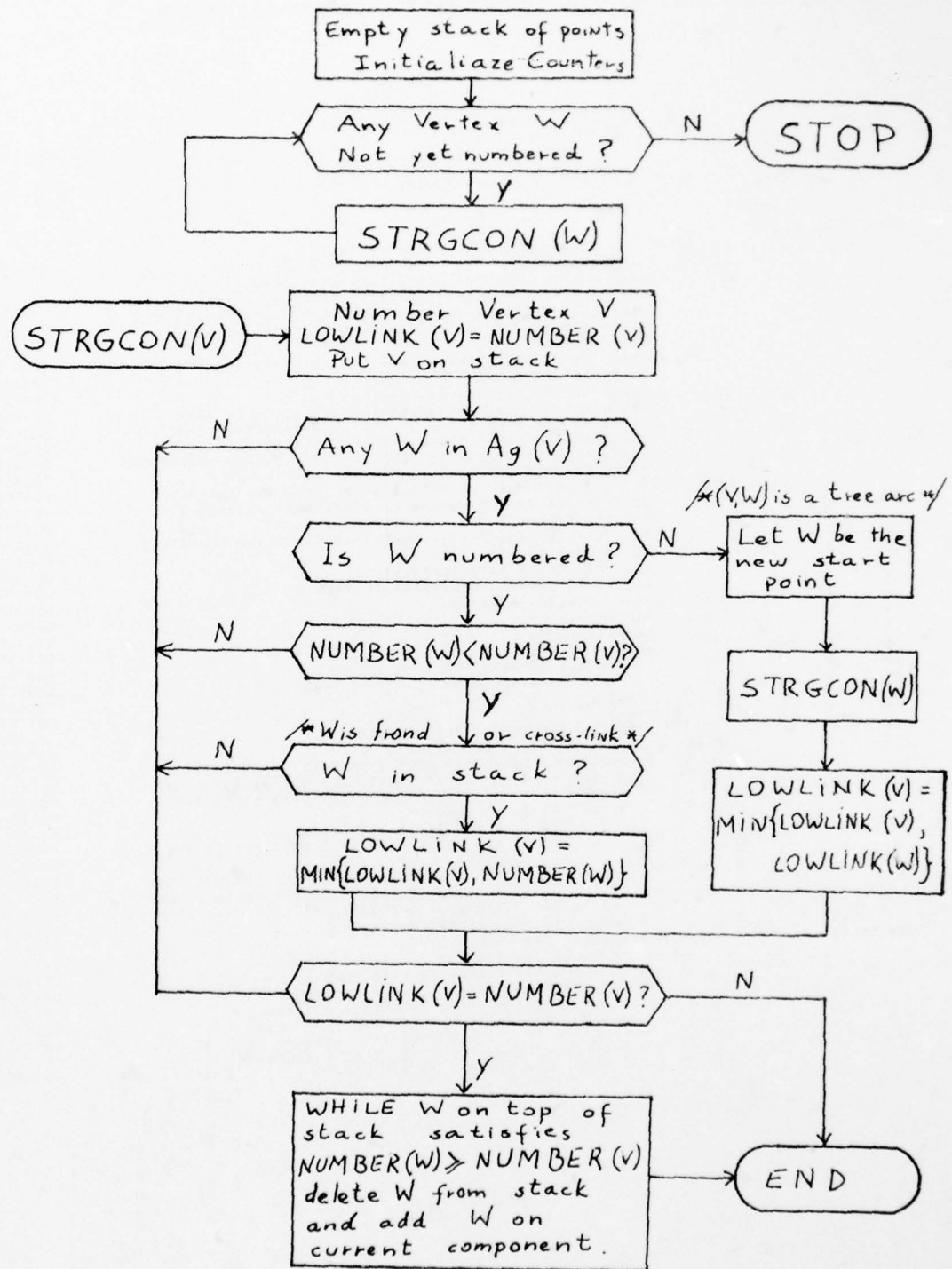


Figure 5.5

Flowchart for Algorithm STRGCON

The following PL/1 program implements Tarjan's algorithm

```

/* GANA 5936 10/12/77 PLIC-VER-12C876 */

STMT LEVEL NEST
1 /* GANA 5476 10/12/77 PLIC-VER-120876 */
CYCLES: PROCEDURE (N);
/* MAIN MONITOR FOR THE "CYCLES ANALYSIS" PROCESSOR.
INITIALIZES VARIABLES, ALLOCATES SPACE AND CALLS
PROCEDURE "STRGCON"; A RECURSIVE DEPTH FIRST SEARCH
ALGORITHM TO FIND STRONGLY CONNECTED COMPONENTS
OF A GRAPH.*/
/* REF: "DEPTH FIRST SEARCH AND LINEAR GRAPH ALGORITHMS"
BY ROBERT TARJAN
PUBLISHED IN SIAM VOL1, NO.2, JUNE 1972.*/
/* THE GRAPH REPRESENTING RELATIONSHIPS BETWEEN DATA
NAMES IS GIVEN BY AN ADJACENCY LIST STRUCTURE "ADJ".
THE STRONGLY CONNECTED COMPONENTS REPRESENT EITHER
CIRCULAR DEFINITIONS OR SETS OF SIMULTANEOUS ASSERTIONS
TO BE GROUPED FOR SOLUTION BY THE SYSTEM.*/
/* "N" IS THE NUMBER OF VERTICES OF THE GRAPH WHICH
CORRESPONDS TO THE NUMBER OF DICTIONARY ENTRIES IN
THE CONTEXT OF "O D E L SYSTEM.*/
DCL (LOWLINK(N), NUMP(N), STACK(N)) BIN FIXED EXT CTL ;
DCL (V, I, PTR) BIN FIXED EXT ;
DCL ONSTACK(N) BIT(1) EXT CTL ;
ALLOCATE LOWLINK, NUMP, STACK, ONSTACK ;
STACK, LOWLINK, NUMP = 0 ;
I, PTR = 1 ;
ONSTACK = "O" B ;
DO V = 1 TO N ;
DO WHILE (NUMP(V) = 0) ;
CALL STRGCON (V, N) ;
END ;
END ;
FREE LOWLINK, NUMP, STACK, ONSTACK ;
STRGCON : PROCEDURE (V, N) RECURSIVE ;
/* PROCEDURE FINDS STRONGLY CONNECTED COMPONENTS
(FRAGMENTS) OF A GRAPH USING A DEPTH-FIRST-SEARCH
METHOD IN A RECURSIVE SCHEMA. THE PARAMETERS ARE
DESCRIBED AS FOLLOWS:
"V" IS A VERTEX OF THE GRAPH
"N" IS THE TOTAL NUMBER OF VERTICES IN THE
GRAPH.*/
/* THE POINTER TO THE ADJACENCY LIST STRUCTURE "ADJ"
IS GIVEN BY FUNCTION "ACWDRA". THE STRUCTURE "ADJ"
GIVES THE PICTURE OF THE INCIDENT NODES AND ADJACENT
NODES OF THE CURRENT VERTEX.*/
DCL 1 ADJ BASED (X),
2 ALLOC BIN FIXED, /* USED ROWS */
2 USED BIN FIXED, /* USED COLUMNS */
2 ENTRIES (A REFER (ALLOC)),
3 COL BIN FIXED, /* COLUMN # */
/* SUCCESSOR COLUMN */
3 CVALUE BIN FIXED, /* COLUMN VALUE */
3 ROW BIN FIXED, /* ROW # */
00040000
00050000
00060000
00070000
00080000
00090000
00100000
00110000
00120000
00130000
00140000
00150000
00160000
00170000
00180000
00190000
00200000
00210000
00220000
00230000
00240000
00250000
00260000
00270000
00280000
00290000
00300000
00310000
00320000
00330000
00340000
00350000
00360000
00370000
00380000
00390000
00400000
00410000
00420000
00430000
00440000
00450000
00460000
00470000
00480000
00490000
00500000
00510000
00520000
00530000
00540000

```


/* GANA 5934 1C/12/77 PLIC-VER-120876 */

STMT LEVEL NEST

```

/* PREDECESSOR ROW */
3 RVALUE BIN FIXED ; /* ROW VALUE */
DCL ACHDRA ENTRY (BIN FIXED)
  RETURNS (POINTER) ;
/* DEFINITIONS:
"LOWLINK(V)" OF A VERTEX V IS THE SMALLEST VERTEX
WHICH IS IN THE SAME COMPONENT AS V AND IS REACHABLE
BY TRAVERSING ZERO OR MORE TREE ARCS FOLLOWED BY AT
MOST ONE "FROND" OR "CROSS-LINK" (AN ARC TO A VERTEX
ALREADY REACHED).

"V" IS THE ROOT OF A COMPONENT IFF LOWLINK(V)=V */
/* THE FOLLOWING ARRAYS ARE MAINTAINED IN THIS
PROCEDURE FOR THE REASONS MENTIONED BELOW :
L O W L I N K : TO KEEP TRACK OF LOWLINK VALUES OF
EACH VERTEX
N L N B : TO NUMBER EACH VERTEX AS IT IS TRACED
S T A C K : TO PUSH THE VERTICES ALREADY TRACED ON
A STACK
ONSTACK : TO VERIFY A VERTEX IS PUSHED ON S T A C K
/* PTR IS A PCINTER USED TO KEEP TRACK OF ELEMENTS
IN S T A C K */
/* "COMP" IS A STRUCTURE USED TO KEEP THE ELEMENTS
WHICH ARE STRONGLY-CONNECTED AS A LINKED LIST */
DCL (LOWLINK(N),NUMB(N),STACK(N)) BIN FIXED EXT CTL ;
DCL X PTR;
DCL V BIN FIXED ;
DCL (J,W) BIN FIXED;
DCL ONSTACK(N) BIT(1) EXT CTL ;
DCL (PTR,I) BIN FIXED EXT ;
DCL 1 COMP BASED(P),
  2 INDEX PIN FIXED,
  2 LINK PTR,
  2 KEY CHAR(2),
  2 STAT BITS BIT(7);
DCL (P,FIRST,LAST) PTR;
DCL CMPCODE BIN FIXED EXT;
X=ACRORA(V) ;
LOWLINK(V),NUMB(V),I=I+1;
/* PUT V ON STACK OF POINTS */
CALL ADD (V,STACK,PTR);
ONSTACK(V) = "1";
IF (X=NULL) THEN GO TO NOADJ;
IF (RUSED=0) THEN GO TO NOADJ;
DO J=1 TO RUSED;
  W=ENTRIES(J).COL ;
  IF (W=0) THEN GO TO SAN ;
  /* W IS IN THE ADJACENCY LIST OF V */
  IF (NUMB(W)=0) /* (V,W) IS A TREE ARC */
    THEN DO ;
    CALL STRGCON (W,N) ;

```

```

00550000
00560000
00570000
00580000
00590000
00600000
00610000
00620000
00630000
00640000
00650000
00660000
00670000
00680000
00690000
00700000
00710000
00720000
00730000
00740000
00750000
00760000
00770000
00780000
00790000
00800000
00810000
00820000
00830000
00840000
00850000
00860000
00870000
00880000
00890000
00900000
00910000
00920000
00930000
00940000
00950000
00960000
00970000
00980000
00990000
01000000
01010000
01020000
01030000
01040000
01050000

```

/* GARA 1039 10/07/77 PLIC-VER-120876 */

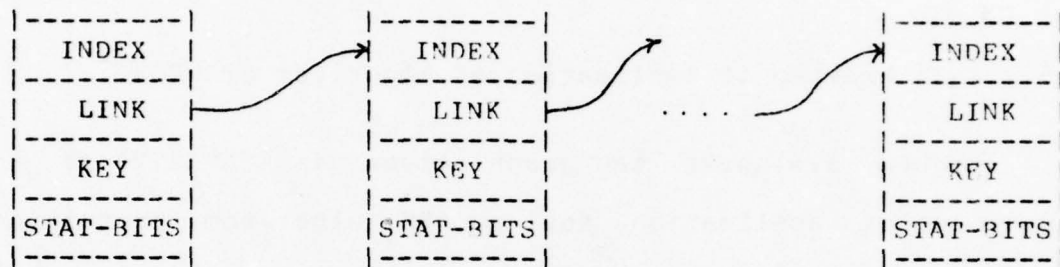
STMT LEVEL NEST

```

42 2 2 LOWLINK(V)=MIN(LOWLINK(V),LOWLINK(W)) ; 01060000
43 2 2 END ; 01070000
44 2 1 ELSE IF ((NUMB(W)<NUMB(V))&ONSTACK(W)) 01080000
/* (V,W) IS A FROND OR CROSS LINK */ 01090000
THEN LOWLINK(V)=MIN(LOWLINK(V),NUMB(W)) ; 01100000
45 2 1 SAN: END ; 01110000
46 2 1 NCADJ: IF (LOWLINK(V)=NUMB(V)) 01120000
47 2 THEN DO; 01130000
48 2 FIRST=NULL; 01140000
49 2 /* V IS THE ROOT OF THE COMPONENT */ 01150000
50 2 /* START A NEW STRONGLY CONNECTED COMPONENT */ 01160000
DO WHILE (NUMB(STACK(PTR))>NUMB(V)&(PTR>0)) ; 01170000
/*DELETE THE POINT FROM TOP OF STACK AND PUT 01180000
IT IN THE CURRENT COMPONENT */ 01190000
ONSTACK(STACK(PTR))='0'B; 01200000
ALLOCATE COMP; 01210000
IF FIRST=NULL THEN FIRST=P; 01220000
ELSE LAST->LINK=P; 01230000
P->INDEX=STACK(PTR); 01240000
P->LINK=NULL; 01250000
LAST=P; 01260000
CALL DELETE (STACK,PTR); 01270000
END; 01280000
IF (FIRST->LINK=NULL) THEN DO ; 01290000
CALL ANALCMP(FIRST); 01300000
CALL PRCOMP(FIRST); 01310000
IF CNPCODE = 0 THEN CALL MRGEASS(FIRST); 01320000
END; 01330000
DO WHILE (FIRST=NULL); 01340000
P=FIRST; 01350000
FIRST=P->LINK; 01360000
FREE P->COMP; 01370000
END; 01380000
73 2 1 END; 01390000
/* INTERNAL SUBROUTINES */ 01400000
74 2 ADD: PROCEDURE (V,STACK,PTR) ; 01410000
75 3 DCL (V,STACK(*),PTR) FIXED BIN ; 01420000
76 3 PTR =PTR+1; 01430000
77 3 STACK(PTR) =V; 01440000
78 3 END ADD; 01450000
79 2 DELETE: PROCEDURE (STACK,PTR); 01460000
80 3 DCL (STACK(*),PTR) FIXED BIN ; 01470000
81 3 STACK(PTR) =0; 01480000
82 3 PTR = PTR-1; 01490000
83 3 END DELETE; 01500000

```

Notice that the DFS algorithm is applied for any non-zero precedence type, that is: to the equivalent binary representation of the adjacency list. The strongly connected elements found by STRGCON are kept as a linked list for further analysis by procedure ANALCMP. The format of this linked list is as follows:



where

INDEX is the dictionary index for the component.

LINK is a pointer to the next component in the list.

KEY is filled by the procedure ANALCMP and identifies the type of element (i.e., assertion name or variable name).

STATUS-BITS are used to further identify the element in case it is a variable. That is if it is resolved as source variable, target variable or interim variable.

Procedure ANALCMP is passed a pointer to this list of strongly connected components, and proceeds to check that:

- i) Elements of strongly connected components are related by precedence types 3 and 4 only (relationships between data names and assertions). If this is not the case, an error message is given to the user.

ii) Assertions must be properly normalized. This is to assume that a variable does not appear both as target and source variable of the same assertion (ex. $A=A+1$).

iii) Finally some other trivial recursive patterns which are known not to have a solution are also reported as errors (ex. the following set of three assertions form a loop without solution: $A=B$; $B=C$; $C=A$).

5.2.4 An Example of Application of Algorithm STRGCON

Figure 5.6 shows the graph given in [TAR 72] as an example of application for a DFS algorithm containing strongly connected components in it.

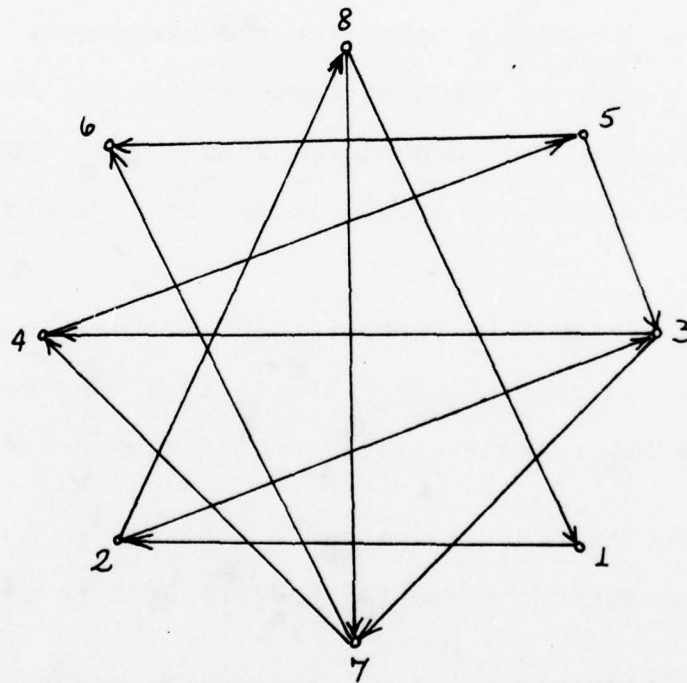


Figure 5.6
Digraph with Strongly Connected Components

The same graph can be considered to represent a mapping of a set of variables onto a set of equations as follows:

```
EQ#1:  VAR#1 = F(VAR#8);
EQ#2:  VAR#2 = F(VAR#1);
EQ#3:  VAR#3 = F(VAR#2,VAR#5);
EQ#4:  VAR#4 = F(VAR#3,VAR#7);
EQ#5:  VAR#5 = F(VAR#4);
EQ#6:  VAR#6 = F(VAR#5,VAR#7);
EQ#7:  VAR#7 = F(VAR#3,VAR#8);
EQ#8:  VAR#8 = F(VAR#2);
```

From this set of assertions, MODEL will generate a precedence matrix of name relationships as shown in the output of Figure 5.7 below

ADJACENCY MATRIX OF NAME RELATIONSHIPS

		5	10	15
1	EQ#1	0	0	0
2	EQ#2	0	0	0
3	EQ#3	0	0	0
4	EQ#4	0	0	0
5	EQ#5	0	0	0
6	EQ#6	0	0	0
7	EQ#7	0	0	0
8	EQ#8	0	0	0
9	VAR#1	0	3	0
10	VAR#2	0	0	3
11	VAR#3	0	0	0
12	VAR#4	0	0	0
13	VAR#5	0	0	3
14	VAR#6	0	0	0
15	VAR#7	0	0	3
16	VAR#8	3	0	0

Figure 5.7
Precedence Matrix Generated for Sample Equations

The equivalent representation of the graph in Figure 5.6 in MODEL is as depicted in Figure 5.8, again square boxes are used to represent assertion names while variable names are encircled.

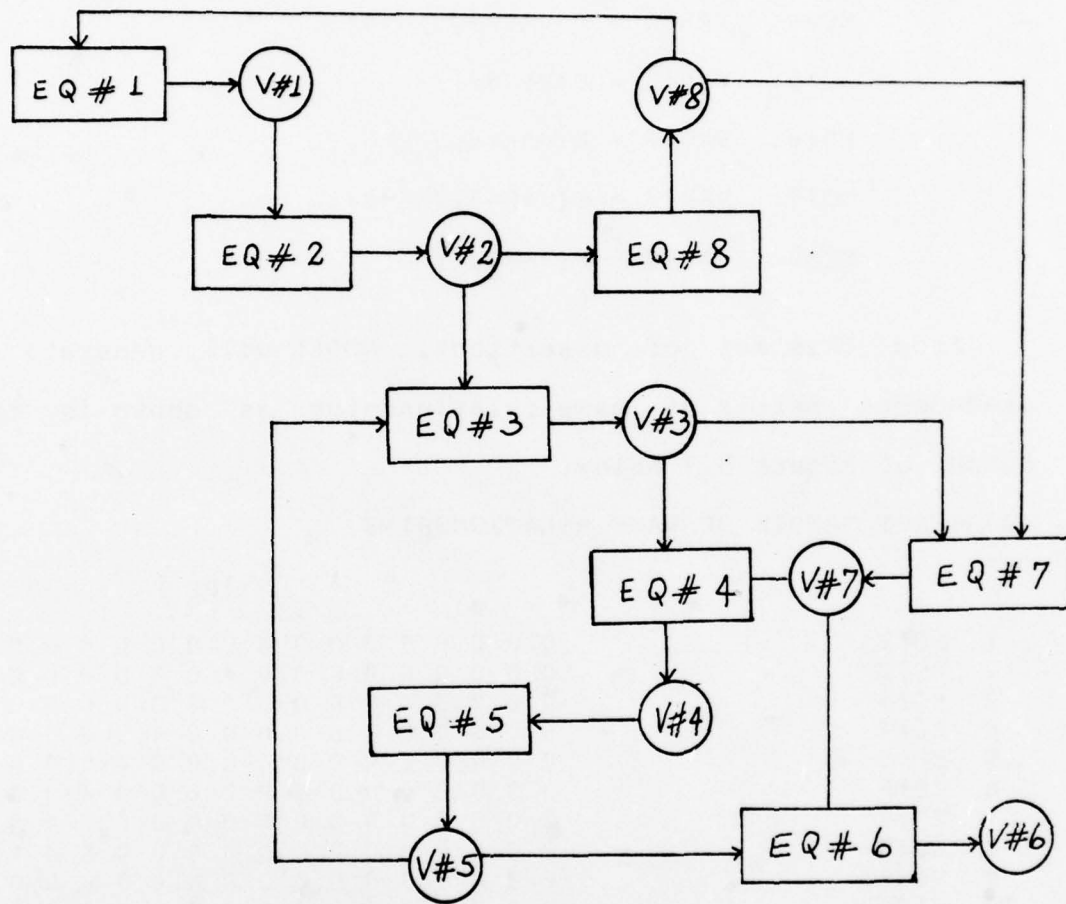


Figure 5.8

MODEL Graphic Representation of Sample Equations

Figure 5.9 illustrate the Jungle generated by the DFS algorithm, with LOWLINK values enclosed in square brackets [] and roots of components marked with asterisk *.

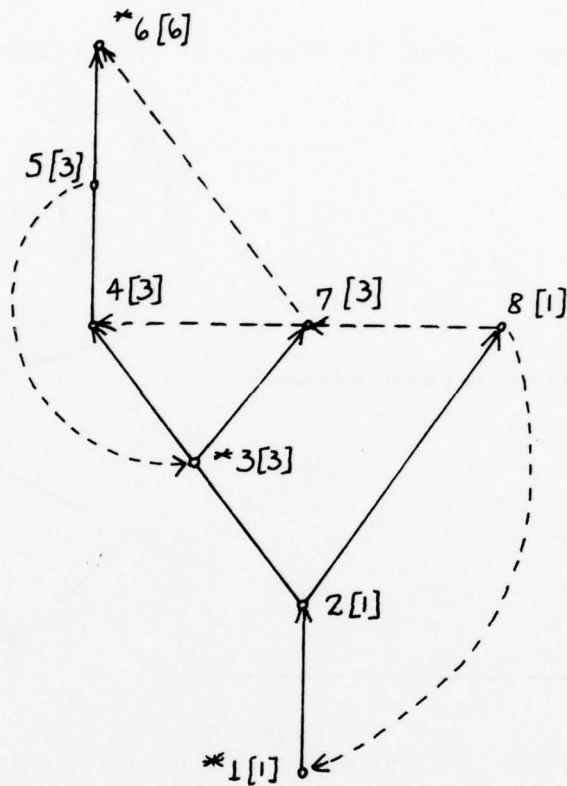


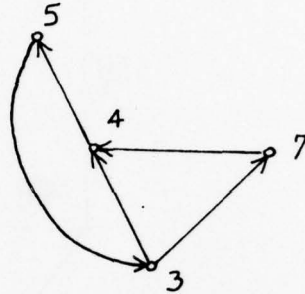
Figure 5.9

Jungle Generated by DFS Algorithm

Finally Figure 5.10 shows the listing of those strongly connected components identified by the algorithm, together with illustrating the interconnection of their elements.

THE FOLLOWING GROUPS OF ITEMS ARE CIRCULARLY DESCRIBED

VAR#7
EQ#7
VAR#5
EQ#5
VAR#4
EQ#4
VAR#3
EQ#3



VAR#8
EQ#8
VAR#2
EQ#2
VAR#1
EQ#1

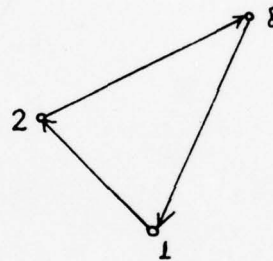


Figure 5.10
Identification of Strongly Connected Components by MODEL

5.3 Linking and Merging Simultaneous Assertions in the Associative Memory (Condensation)

Those assertions which belong to the same strongly connected component and which are resolved as a simultaneous set by the cycles analysis are linked together in the associative memory. This process can be considered as the equivalent of the "condensation" of nodes in a digraph to transform it into an acyclic structure. By linking the assertion together it is possible to access the whole group when interrogating the associative memory. Also all the information pertinent to an individual assertion in the group is still kept for purposes of documentation, code generation and further analysis by MODEL. Similarly those assertions which have a simple name in common are also considered to be a set of simultaneous assertions and are linked in the same way in the associative memory (for instance, the assertion names COUNTRY.GNP and COUNTRY.C have in common the simple name COUNTRY and therefore are linked together using the same logic as in the case of simultaneous assertions analyzed by the cycles processor).

Whenever two or more assertions are given the same name, then those assertions are also treated as simultaneous and their structural information again linked, but this time under the same storage entry name. This process will be referred as "merging" the assertions.

Since during the syntax analysis each statement is

stored in the associative memory in a coded form, next sections describes the structure and interrelation between elements of an assertion storage entry.

5.3.1 Directory Structure

The directory uses a "branch and bound" binary tree structure for efficient searching and maintenance of its entries. That is, when generating the directory the first key name becomes the root of the tree while the next key is entered above or below it in the tree by lexicographic order. Every key name have one entry in the directory and each entry points to to the first storage entry containing that name. A linked list is then maintained from this first storage entry to other storage entries containing the same key name.

Each directory entry has the following format:

```

|-----+-----+-----+-----|
|KEY-NAME|UP-PTR|DOWN-PTR|F-PTR|
|-----+-----+-----+-----|

```

where

KEY-NAME is a string of up to 10 characters (padded with blanks);

UP-PTR and DOWN-PTR are pointers to other directory entries in the tree structure, whose key names are up or down respectively, in the lexicographical sense;

F-PTR is a pointer to the first storage entry that contains the key name.

5.3.2 Key Names

The names given by the user for naming data, assertions, etc., are prefixed by a two character code which identifies the type of name being stored. This compound name becomes the Key name by which is possible to retrieve information later on. If the data name supplied by the user is longer than 8 characters, then it is encoded into a 4 character length string which is then stored in the directory after prefixing by the appropriate two character code. Those names which are longer than 8 characters are saved in a tree structure, and the coded string in the directory is actually an index to the name in the tree structure.

The complete list of identification prefixes can be found in the MODEL system documentation [SHA 78], for example data names representing GROUPS use the prefix 'SG', assertions use the prefix 'SA' and so on. In addition, the first key name in any storage entry is a special two character name which identifies the type of statement the storage entry represent. In the case of assertions, the first key name of the storage entry must be 'SA'

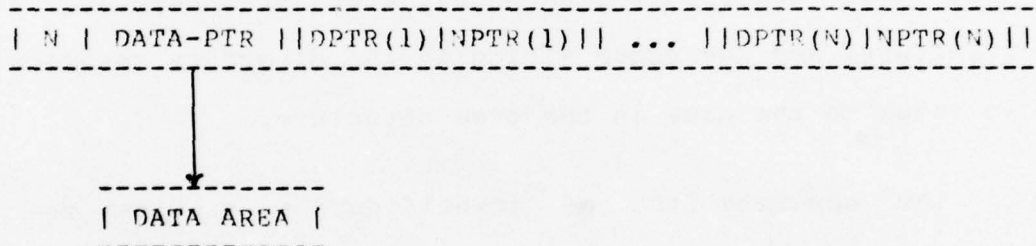
5.3.3 Storage Entries for Assertions

One storage entry is created for each MODEL statement, in general the storage entries consists of two parts:

- (1) Key names to be entered in the directory; and
- (2) Auxiliary data from the source language statement.

This data is in encoded form and it is not used as basis for retrieval.

The storage entries have the following format:



where

N is the number of key names in the storage entry;

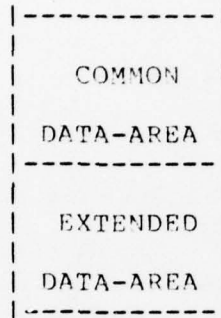
DATA-PTR is the pointer to the auxiliary encoded representation of the source statement (data area);

DPTR(I) (I=1 TO N) is the directory entry pointer to the ith key name; and

NPTR(I) (I=1 TO N) is a pointer to the next storage entry with the same key name.

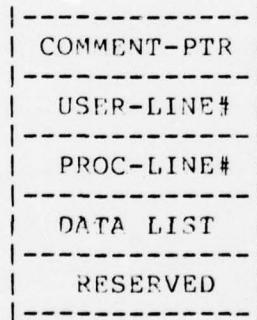
The DATA AREA consists of two parts:

- (1) Common data area and
- (2) Extended data area



5.3.3.1 The Common Data Area

The common data area is used by all types of statements and contains five entries with the following format:



where

COMMENT-PTR points to the "comment string" associated with the statement;

USER-LINE# is the input line number in which the statement begin;

PROC-LINE# is a statement number supplied by the system;

DATA-LIST is a pointer to a block of data which is filled during the creation of the dictionary.

5.3.3.2 The Extended Data Area for Assertions

The extended data area contains additional information for different groups of statements. The format of the extended data area for assertions is as follows:

VL-PTR
ASS-PTR
NASS-PTR
SV-PTR
TV-PTR
FN-PTR
INIT-PTR
TEST-PTR
SOL-PTR

where

VL-PTR points to a list of variables used in the assertion.

This pointer is obtained during the creation of the dictionary.

ASS-PTR points to the root of the "derivation tree" (DT) of the assertion. This derivation tree is generated during the syntax analysis phase. The DT is a list of entries that represent each production in the EBNF for the assertion (see [SHA 78] for detailed description

of the list).

NASS-PTR points to the data area or to the storage entry of another assertion if the two assertions belong to a set of simultaneous equations.

SV-PTR and TV-PTR are pointers to the source and target variable list, respectively, for this assertion. The list is specified by the user in the "header section" of the assertion.

FN-PTR is a pointer to a list of function names referenced in the assertion.

INIT-PTR is a pointer to a list of initial values for variables used in assertions. These initial values are used as the starting point for iterative solution schemes of simultaneous assertions.

TEST-PTR is a pointer to a list of conditions associated with each variable for termination of the iterations in the solution of simultaneous equations by iterative methods. The list represent the convergence criteria.

SOL-PTR is a pointer to a list of solution methods and associated parameters for simultaneous assertions.

5.3.4 The LINKAS procedure

The procedure LINKAS(PTR) is passed a pointer (PTR) to the linked list of strongly connected components representing elements of a simultaneous assertion group. It then proceeds to retrieve and link the storage entries corresponding to assertions in the group. The linkage is done by setting the pointer NASS-PTR in the extended data area of the assertion storage entry. NASS-PTR actually points to a structure with the following format:

```

|-----+-----+-----|
| CODE | FIRST-PTR | NEXT-PTR |
|-----+-----+-----|

```

where

CODE is 'D' or 'S' or blank and represents the type of pointer specified by NEXT-PTR.

FIRST-PTR points to the first storage entry in the linked list.

NEXT-PTR is a pointer to the data area of an assertion (if CODE = 'D') or to another storage entry (if CODE = 'S'). If the code is blank, then the assertion represented by the current data area is the last assertion in the set.

Since LINKAS is called at the end of cycles analysis to link the storage entries of a simultaneous set of assertions, it always set the code to 'S' or blank. After processing of the storage entries by LINKAS, only the entry for the first assertion is kept in the "precedence matrix"

list.

Assertions using name qualification and which have a simple name in common are linked in a similar way but before cycles analysis, during the syntax analysis stage. If the user gives exactly the same name to a group of two or more assertions, these are "merged" together also during the syntax analysis phase by creating only one storage entry for the common name, but this storage entry will contain a linked list of extended data areas, one for each assertion in the group. The CODE field in the structure pointed by NASS-PTR in this case will be 'D'. The whole group of assertions with the same name are considered as one single node in the graphic representation (one entry in the adjacency list).

5.4 Tearing Strongly Connected Components

The partition induced into a system by the identification of the strongly connected blocks is essential for an automatic program generator and also aids in understanding and solving large and complex models. However in many instances (for example in econometric studies and in models of chemical process) the size of the simultaneous blocks might be so large that still it will be difficult to trace possible errors and to understand the interrelationships between the components of the block. It is then legitimate to ask whether it is possible to eliminate from the adjacency list one or more elements in the group such that it then becomes feasible further decomposition into smaller and more manageable blocks. This process of removing elements from the original structure is called "Tearing". Steward [STE 65] proposed an algorithm based on elementary circuits inside each block. Steward's algorithm is based solely on structural information and not on analysis of how the variables to be removed (torn element) are used in the equation. Later Soylemez [SOY 71] added symbolic algebraic analysis to Steward algorithm and used it to generate Fortran solution programs for chemical models.

Tearing will normally imply an iteration scheme in which the torn element is given an initial value so that the remaining blocks can be solved independently. The solution

of these blocks in turn might determine a new value for the torn element. This process is continued until an overall convergence criteria is met. Normally the initial value for the torn element is set equal to the observed or historical data. If a sensitivity analysis shows that the torn element does not affect substantially the solution of the remaining equations, it is then possible to set the torn element to a constant and proceed to the study and solution of the remaining elements of the system as if it were decomposable or causal. This problem is similar to the "semidecomposition" problem found in the study of sequential machines [HAR 66]. Sufficient literature is found today on the algebraic properties of these machines [KRO 62] and conditions for the realization of decomposition (cascade or parallel), such as to provide a solid theoretical framework for further studies on their relation and application to dynamic models.

To illustrate this concepts consider the adjacency matrix in Figure 5.11. This matrix represent a system which is undecomposable.

	1	2	3	4	5
1	0	1	0	0	0
2	0	0	1	1	0
3	1	0	0	0	0
4	0	0	1	0	0
5	0	0	0	1	0

Figure 5.11
Undecomposable Adjacency Matrix

A transition graph can be drawn from this matrix as represented in Figure 5.12.

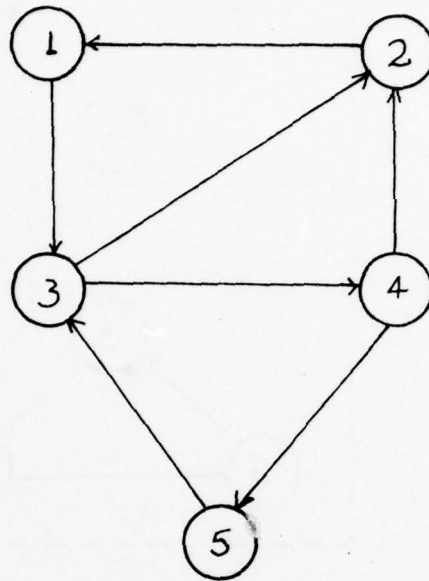


Figure 5.12

Transition Graph

Lets consider the partitions 1,2,3 and 3,4,5, and consider these to be two blocks. Then the same transition matrix can be represented by the graph in Figure 5.13.

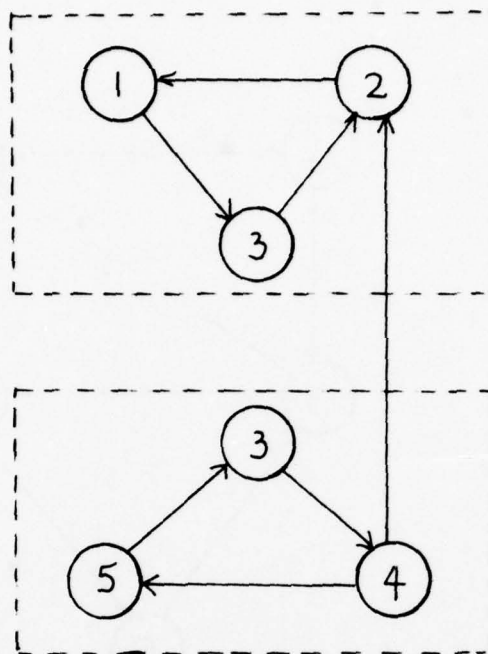


Figure 5.13

Semidecomposition Illustration

If the matrix were representing a simultaneous system of equations, an interpretation would be that the solution can be found by assigning a value to variable 3, then solving for variables 4 and 5. Variable 2 can then be determined and 1 from 2. From the values found for variables 5 and 1, a new guess of variable 3 is determined, and so on.

For example, in describing the LINK world trade model [BAL 73, KLE 76] (refer also to chapter 7 of this dissertation), semidecomposition is implicit by a priori partition of the

totality of the model into national models and a trade model. Also the LINK algorithm defines the torn elements to be import prices (PM) and export Volumes (X), used as exogenous in the country models and later computed endogenously by the trade model. Total world trade (TW) is also used initially exogenous with the country best guess and computed later from the account identity

$$TWS(t) = \sum_{i=1}^n XSi(t)$$

The main objections to Steward's algorithm had been the computational complexity of tracing every elementary loop of a big system of simultaneous equations. However with the advent of efficient linear enumerating algorithms [JOH 75] it becomes a feasible, but still time consuming, proposition.

A totally different approach was experimented with econometric models as part of this dissertation research, and reported in some detail in [GAN 76c]. Basically it was realized the similarities between the problem of breaking a simultaneous system of equations into groups of equations which are in some sense "closely" related to other equations in the group but "loosely" related to the ones in the other groups, and that of grouping "similar" or "like" records or descriptors together in information retrieval and automatic classification systems. These systems use clustering techniques in order to facilitate retrieval and reduce access time. In particular a very efficient heuristic

algorithm (CLASFY [LIT 69]) was applied to an econometric model of Germany to study the feasibility of its decomposition as well as its convergence properties. More experimentation and research is needed before asserting definitive conclusions on the use of these techniques. What appears to be valuable in CLASFY is that it is possible to define the number of blocks of the partition as well as sensitivity levels or thresholds of association.

In summary, the tearing of big simultaneous systems is an area which deserves attention, particularly in light of the advent of low cost microprocessors and the possibilities of parallel computation. Also the addition of symbolic algebraic analysis to an efficient algorithm such as CLASFY might prove to be a valuable tool for understanding complex systems.

CHAPTER 6

Code Generation and Solution Procedures

6.1 Introduction

After a given specification is analyzed for completeness and cycles or simultaneous assertions, its statements are sequenced and analyzed for optimal scope of iterations. A flowchart table containing the complete control logic for the desired program is then generated and used in the subsequent code generation stage. In general the generation of PL/I code from the flowchart table is straightforward and its implementation for a previous version of MODEL was reported by Rin [RIN 76]. This chapter extends that methodology to include solution procedures for sets of simultaneous assertions (possible non-linear), and discusses some theoretical and design aspects for its implementation. Considerations are also given for the case of assertions which are not "normalized", i.e., without a single one-variable dependent term on the left of the equal sign. For example the following assertion can be used in describing a module:

$$\begin{aligned} \text{VX(I)/PX(I)} &= \text{A(I)} + \text{B(I)*SUM(A70(I,J)*VM(J),J)/PX(I)} \\ &+ \text{G(I)*PCOM(I)/PX(I)} + \text{D(I)*T;} \end{aligned}$$

where the LHS expression contains two variables, VX and PX. Model builders can take advantage of this facility and generate modules with different choices of normalization when a simultaneous block presents convergence problems. An

algorithm is given to automatically produce a "normalized" assertion once the user selects a target variable.

The solution procedures are transparent to the user, but for their associated set of parameters. They are generated independently for each block of simultaneous assertions present in a module. In summary, computational efficiency is achieved through three specific features of MODEL implementation:

- 1 - The sequencing phase of MODEL produces an ordering of the statements for their evaluation in the ultimate program. The resulting adjacency representation of a module becomes a "block triangular" structure. The flowchart can be obtained after this phase, once the scope of iterations have been determined for maximum efficiency.

- 2 - A module is broken down into as many recursive blocks of simultaneous equations as possible, then during execution of the generated PL/1 program, each block is solved separately. This technique requires less computer time and program memory than would be needed for solution of the entire set of assertions in the module as a single unit.

- 3 - Further optimization at the machine instructions level can be achieved by compiling the target PL/1 program with the optimizing compiler. This particular

phase mechanizes a procedure used by many simulation programs [JOH 76] where "constant" terms or expressions which are part of an assertion that is iterated for simultaneous solution are placed in a separate subroutine for single evaluation. In a large scale dynamic model, these constant terms can be expressions of lagged variables, exogenous variables, coefficients or disturbances. Since these expressions do not change from iteration to iteration, great savings in computation time are obtained when they are transferred outside the iterative solution loop. Other optimization features at this level include the elimination of common and redundant expressions, as well as simplification of expressions (see the IBM manual [PL1 75] for general information on the PL/1 optimizing compiler).

6.2 Solving Non-linear Simultaneous Equations

While there exist many methods available for solving a set of general non-linear algebraic equations, the discussion in this chapter will center only on the implementation of the Gauss-Seidel and Successive Over Relaxation (SOR) techniques. These methods are widely used in econometric research for simulation and forecasting purposes, but they are not recommended for application areas whose models are characterized by strong non-linearities and where a good starting approximation for the solution is not

generally known.

Since most econometric models are close to linear around the solution neighborhood, and it is possible to use the previous period observed or computed values as a good initial solution estimate, the economist can take advantage of the conceptual simplicity of the Gauss-Seidel and SOR methods as well as of their lower implementation cost. For application in the engineering or biological sciences, the same implementation guidelines can be followed by introducing solution procedures such as the full Newton's method.

6.2.1 General Principles

To describe the solution procedures consider a block of simultaneous assertions as follows:

$$(i) \quad Y = F(Y, X)$$

where Y is an n vector of target (endogenous) variables $(Y_1, \dots, Y_n)'$, F is an n vector of functions $(f_1, \dots, f_n)'$, and X is an m vector of source variables $(x_1, \dots, x_m)'$. Note that in the case of a dynamic econometric model, for example, X will include all predetermined exogenous and lagged variables plus the parameters and disturbances. For purposes of stating the iterative algorithm, it is convenient to suppress the source elements in the notation and use the compacted form for the system:

$$(i)' \quad Y = G(Y)$$

where G is the n vector of functions $(g_1, \dots, g_n)'$. This

representation assumes that each assertion has been normalized in such a way as to determine a distinct endogenous variable for the system.

Suppose now that Y^0 is some initial approximation vector to the solution of the system. For a dynamic system this can effectively be assumed as $Y_t^0 = Y_{t-1}$. Define iteratively

$$(ii) \quad Y^{k+1} = G(Y^k) \quad k=0,1,2,\dots$$

The sequence $Y^0, Y^1, Y^2, \dots, Y^k, Y^{k+1}$ will converge to a unique solution (fixed point) between a given interval $[a,b]$ provided that G satisfies the well known conditions stated by the contraction mapping theorem. In general these conditions are either not fulfilled by some large scale models, or very difficult to come by analytically. Fortunately in most econometric models convergence and uniqueness can be expected in practice because of their property of local near-linearity and the fact that the starting values $Y_t^0 = Y_{t-1}$ are very good approximations to the solution. In fact in cases where multiple solutions have been encountered, they normally present an unrealistic scenario and can be discarded (for example negative employment rates [FEI 71]).

The iterative procedure given by equation (ii) is known as the Jacobi method. The Gauss-Seidel technique takes advantage of existing sequential processors and substitutes the most recently computed value for endogenous variables y_1, \dots, y_{i-1} into the right hand side of equation for y_i .

References to variables y_{i+1}, \dots, y_n still use the previous iteration approximation. That is;

$$\begin{aligned} y_1^{k+1} &= \sigma_1(y_2^k, \dots, y_n^k) \\ y_2^{k+1} &= \sigma_2(y_1^{k+1}, y_3^k, \dots, y_n^k) \\ &\cdot \\ &\cdot \\ &\cdot \\ y_n^{k+1} &= \sigma_n(y_1^{k+1}, \dots, y_{n-1}^{k+1}) \end{aligned}$$

By updating the solution vector as the new values are calculated and not at the end of each iteration, this process becomes computationally more efficient than the Jacobi method (a different analysis will be required if the methods were implemented in certain classes of parallel processors).

6.2.2 Selecting the Output Set

Considerations must be given to the normalization rule used for each assertion. Most economic models show a "natural" or logical normalization for each equation, however difficulty arises when there is a reasonable choice for normalization (selection of the output set) among a group of assertions. In fact the natural normalization used for estimation of the model need not be the same used for solution of the model. These points can be illustrated using a simple two equation linear system;

$$(1) \quad v_1^{k+1} = ay_2^k + u_1$$

$$(2) \quad v_2^{k+1} = by_1^{k+1} + u_2$$

The solution to this system is found by inspection to be

$$y_1 = (au_2 + u_1)/(1 - ab)$$

$$y_2 = (bu_1 + u_2)/(1 - ab)$$

in terms of the Gauss-Seidel sequence of iterations:

$$(3) \quad v_1^{k+1} = (ab)^{k+1} v_1^0 + \sum_{i=0}^k (ab)^i (u_1 + au_2)$$

which in turns can be substituted in (2) to compute y . For the sequence in (3) to converge the condition is:

$$|ab| < 1$$

Figure 6.1 illustrates a convergent solution path.

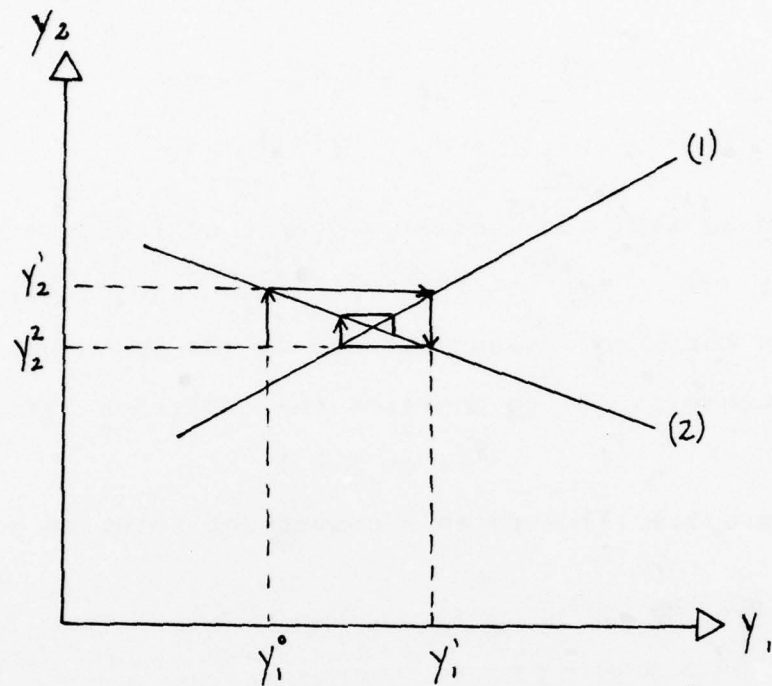


Figure 6.1

Convergent Solution

Suppose now that the opposite normalization is selected, that is

$$\begin{aligned}
 (1)' \quad y_1^{k+1} &= (1/b) y_2^k - (u_2/b) \\
 (2)' \quad y_2^{k+1} &= (1/a) y_1^{k+1} - (u_1/a)
 \end{aligned}$$

then the sequence

$$(3) \quad y_1^{k+1} = (1/ab)^{k+1} y_1^0 - \sum_{i=0}^k (1/ab)^i (u_1/ab - u_2/b)$$

will diverge for $|ab| < 1$ as illustrated in Figure 6.2.

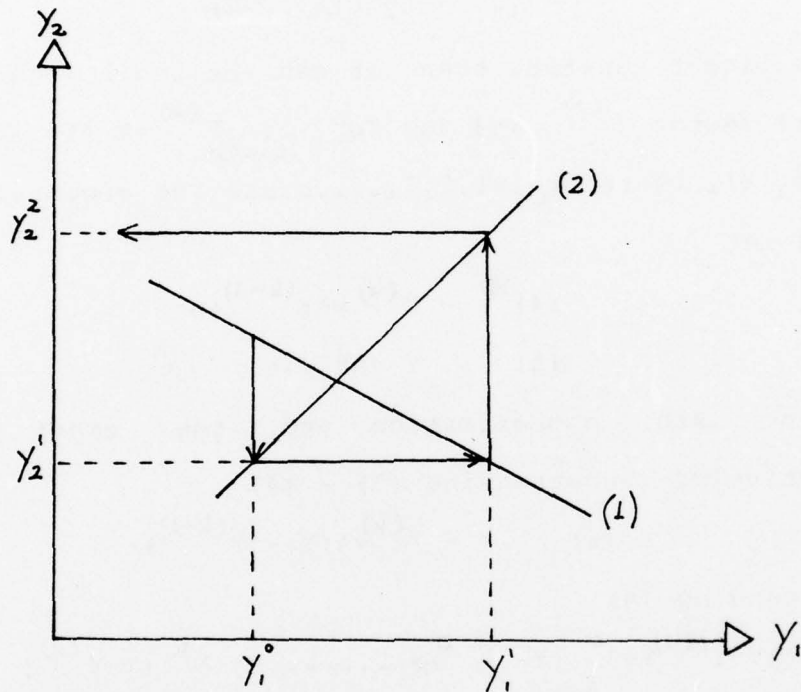


Figure 6.2

Divergent Path

Section 6.4 will present some mechanisms by which a user can experiment with different normalization rules. In

general the model builder should study and follow the logic of the system under study for its normalization.

6.2.3 Necessary and Sufficient Conditions for Convergence

Given the Jacobi process of Successive approximations, which is defined by

$$(4) \quad Y^{(k)} = AY^{(k-1)} + b$$

for a linear system, then it can be shown that for any initial vector $Y^{(0)}$ and any "b", $\lim_{k \rightarrow \infty} Y^{(k)} = Y$ if and only if all $|\lambda_i| < 1$, where λ_i $i=1,2,3,\dots,n$ are the eigenvalues of A .

Proof: Let

$$(4) \quad Y^{(k)} = AY^{(k-1)} + b$$

$$(5) \quad Y = AY + b$$

be the k th approximation and the exact solution respectively. Subtracting (5) - (4)

$$(6) \quad Y - Y^{(k)} = A(Y - Y^{(k-1)})$$

and expanding (6)

$$Y - Y^{(k)} = A(Y - Y^{(k-1)}) = A^2(Y - Y^{(k-2)}) = \dots = A^k(Y - Y^{(0)})$$

$$\text{therefore } \lim_{k \rightarrow \infty} (Y - Y^{(k)}) = \lim_{k \rightarrow \infty} A^k (Y - Y^{(0)})$$

for convergence it is necessary that $\lim_{k \rightarrow \infty} (Y - Y^{(k)}) = 0$, hence the necessary and sufficient condition is $\lim_{k \rightarrow \infty} A^k = 0$, or all eigenvalues of the matrix A must be less than unity in absolute value.

In order to state the conditions for a linear system using Gauss-Seidel, let the general system of algebraic equations be given as

$$(7) \quad BX = C \text{ with } |B| \neq 0$$

then the necessary and sufficient conditions for convergence of this method is that all the eigenvalues of the matrix $B_1^{-1} \cdot B_2$ be less than unity in absolute value, where B_1 is the upper triangular matrix of A , including the main diagonal, and B_2 is the lower triangle matrix with its diagonal = 0.

Proof: for (7) the Gauss-Seidel procedure is

$$B_1 Y^{(k)} + B_2 Y^{(k-1)} = C$$

or

$$(8) \quad Y^{(k)} = -B_1^{-1} \cdot B_2 Y^{(k-1)} + B_1^{-1} \cdot C$$

therefore by setting $A = -B_1^{-1} \cdot B_2$ and $b = B_1^{-1} \cdot C$, (8) becomes as (4) from which we know from Jacobi iteration conditions that all eigenvalues of A must be less than unity in absolute value.

Unfortunately, the implementation of these necessary and sufficient conditions in a system are inefficient since first the eigenvalues of the matrix are not known in advance, and their computation will be required before generation of the solution procedure. Therefore a simple sufficient only criteria for the Gauss-Seidel method will be given which can easily be implemented [FAD 65]. The Gauss-Seidel procedure will converge if

$$\max_i \sum_{j=1}^n |a_{ij}| < 1$$

Similar sufficient conditions can be stated for sets of non-linear equations, however they require computation of all partial derivatives, and knowing an interval about the root. These are major problems for automatically checking convergence characteristics before generation of a solution

procedure.

6.2.4 An Improved Method

Another factor to consider when solving a simultaneous block of equations is the ordering of the equations inside the block. It is known that different ordering may have a dramatic effect in the speed of convergence for the Gauss-Seidel method. Several schemes to improve Gauss-Seidel in this respect have appeared in the literature under the generic name of relaxation methods. One of them determines the order of execution of an equation according to the size of the residuals. This method is good for hand calculations but very cumbersome to program. A widely used improvement known as the Successive Over Relaxation (SOR) method is obtained by modifying the step size at each iteration adjustment as follows:

$$y_i^{k+1} = h\hat{y}_i^{k+1} + (1-h)y_i^k \text{ or}$$

$$y_i^{k+1} = y_i^k + h(\hat{y}_i^{k+1} - y_i^k)$$

where \hat{y}_i^{k+1} is the function value of the i th target (endogenous) variable in the current iteration and h is the relaxation parameter to be chosen by experimentation between 0 and 2. The traditional Gauss-Seidel method results when $h=1$. This parameter can also be varied in the computational process itself to bring convergence to the final solution faster.

More detailed analysis of these and other methods, and

of their convergence properties can be found in standard numerical analysis textbooks (see for example [DAH 74] and [ORT 70]).

6.3 Generation of a Solution Procedure

6.3.1 Simultaneous Assertions in the Associative Memory

A group of simultaneous assertions may be either "merged" into a single storage entry in the AM, if the user provides the group with a common assertion name, or the storage entries corresponding to distinct assertion names "linked" as a consequence of the cycles analysis or a common qualifier in the name supplied in their description. Recalling the format of a storage entry, as described in chapter 5 section 5.3.3, the resulting structure in the AM becomes a recursive linked list. Assume a set $A = \{A_1, A_2, \dots, A_n\}$ of assertions in a simultaneous group, a set $S = \{S_1, S_2, \dots, S_m\}$ of storage entries where $m \leq n$ equals the total number of distinct names used for assertions in A , and finally a set $D = \{D_1, D_2, \dots, D_n\}$ of data areas corresponding to each assertion A . Then the list can be formally specified as follows:

$$i) A_i, A_j \in S_k \Leftrightarrow \text{NAME}(A_i) \equiv \text{NAME}(A_j)$$

$$ii) D_i \rightarrow D_j \Leftrightarrow A_i, A_j \in S_k$$

$$iii) D_i \rightarrow S_k \Leftrightarrow \text{NAME}(A_i) \neq \text{NAME}(A_j \in S_k) \\ \& \text{NAME}(A_i) \cap \text{NAME}(A_j) = \langle \text{qname} \rangle \\ | \text{STRONG_CONNECT}(A_i, A_j)$$

where the symbol " \rightarrow " indicates a pointer relationship and $1 \leq i, j \leq n$ and $1 \leq k \leq m$. Note that in ii) and iii) the pointer relationship (\rightarrow) may imply an ordering if the condition $i < j$ is added. This ordering could be used to indicate a particular execution sequence in the solution procedure for critical systems of equations. For implementation of such alternative, the ordering should be considered as a vector of priorities associated with A_i , which could be automatically generated by an algorithm such as the relaxation method based on residuals, or defined as a parameter by the user. In this form MODEL will maintain its non-procedural characteristic at the statement level.

Figure 6.3 illustrate a list in the AM for a general group of simultaneous assertions.

$$\mathcal{A} = \{A_1, A_2, \dots, A_n\} \xrightarrow{1-1} \mathcal{D} = \{D_1, D_2, \dots, D_n\}$$

$$\mathcal{S} = \{S_1, S_2, \dots, S_m\}$$

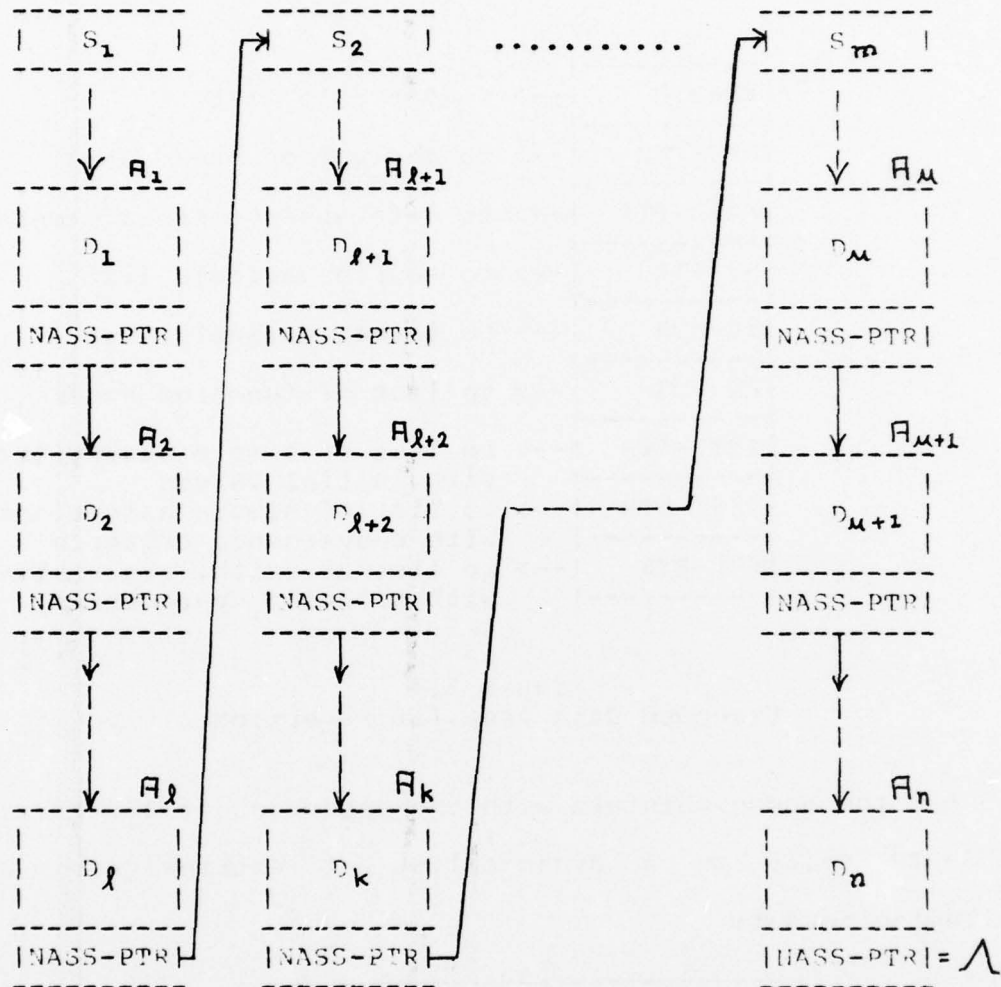


Figure 6.3
Simultaneous Assertions in the AM

The data areas (D_i) will contain pointers to each assertion and their associated solution parameters, as depicted in Figure 6.4.

D_i	-----	
	VL-PTR	--> to variable list

	ASS-PTR	--> to derivation tree for A_i

	NASS-PTR	--> to data area or storage entry

	SV-PTR	--> to source variable list

	TV-PTR	--> to target variable list

	FN-PTR	--> to list of function names

	INIT-PTR	--> to list of simple assertions
	-----	with initial values
	TEST-PTR	--> to list of simple assertions
	-----	with convergence criteria
	SOL-PTR	--> to list of arith. expressions
	-----	with solution parameters

Figure 6.4
Extended Data Area for Assertion

All the above pointers with the exception of ASS-PTR and NASS-PTR point to a dynamic list of entries with the following format:

```
-----
| N | PTR(1) | .... | PTR(N) |
-----
```

where N is the number of elements in the list and $PTR(I)$, $I=1$ to N are the pointers to assertion components as specified in Figure 6.4.

The pointer NASS-PTR used for the linked list has the following format:

```
-----
| CODE | FIRST-PTR | NEXT-PTR |
-----
```

where CODE = D | S | blank

FIRST-PTR points to the storage entry of the first assertion in the group, and

NEXT-PTR points to the data area if CODE = D, to a storage entry if CODE = S or is the last assertion in the set if CODE = blank.

6.3.2 Algorithm SOLVEGS

SOLVEGS will be invoked each time a node in the flowchart table represents a group of simultaneous assertions. It will traverse the linked list and generate a solution procedure for the set. The simplest implementation could be a single overall iterative method for every assertion, merged or linked, which is a member of the list. While such implementation is perfectly acceptable and will satisfy most applications, the design presented below fully utilizes the structure of the list allowing nested generation of solution procedures up to a depth of two. This technique gives more semantic power to MODEL and allows broader experimentation alternatives. In fact it is not difficult to extend the generation algorithm in order to use the complete nested capabilities provided by the "tree" structure of qualified names, and thus permit inner

AD-A066 192

MOORE SCHOOL OF ELECTRICAL ENGINEERING PHILADELPHIA P--ETC F/G 9/2
USE AND EXTENSION OF AN AUTOMATIC PROGRAM GENERATOR FOR MODEL B--ETC(U)
OCT 78 J L GANA

N00014-76-C-0416

NL

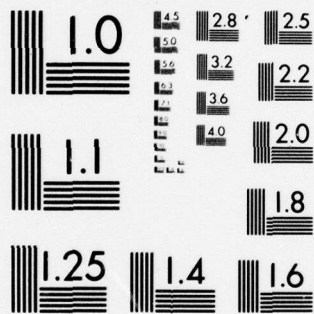
UNCLASSIFIED

78-02

4 OF 6

AD
A088152





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

iteration of selected subgroups at any recursive depth.

The global solution level is composed of those assertions identified by the "cycles" processor as well as those with a common qualifier name, while the inner or second level will consist of every subsystem using the same assertion name. Thus SOLVEGS works as follows:

- 1.- It will generate a procedure call to the global solution procedure, and the corresponding procedure code for each list of storage entries \mathcal{D} .
- 2.- Each time a storage entry $S(i)$ is found containing a list of data areas $D(j)$, $j > 1$, it will generate a second level procedure call together with the solution code for the list $\mathcal{D}(\ast)$.

The process is illustrated with examples in Figure 6.5. Each example shows in the left hand column the assertion names and solution parameters (method, maximum number of iterations) as a representation of different structures of the list in the AM. The right hand side gives the corresponding code generation schema.

* Equivalently the second level could be generated as an inner block with solution code for \mathcal{D} and delimited by BEGIN - END statements.

A1:SOL:G_S,40:<assertion>;		CALL A1;	
A2: <assertion>;		A1:PROC;	
: :		SOL { \mathcal{A} code RETURN;	G_S,40
: :			
An: <assertion>;			

Figure 6.5a
Code Generation: One Storage Entry for Each Assertion

A1:SOL:G_S,40:<assertion>;		CALL A1;		
: :		A1:PROC;		
AK:SOL:G_S,10:<assertion>;		SOL { \mathcal{A} -AK code CALL AK RETURN;	AK:PROC	
AK: <assertion>;	SOL G_S 40			SOL { AK code RETURN;
: :				
AK: <assertion>;		10		
AK+1: <assertion>;		END A1;	END AK;	
: :				
An: <assertion>;				

Figure 6.5b
Code Generation: Subgroup Under Single Storage Entry

A1.X:SOL:G_S,50:		CALL A1;		
G_S,20:<assertion>		A1:PROC		
A1.X: <assertion>		SOL { CALL A1_X CALL A1_Y code RETURN;	A1 X: PROC	
: :				SOL { A1_X code RETURN;
: :				
A1.X <assertion>	SOL G_S 50	20	END A1_X;	
A1.Y:SOL:G_S,50:		END A1;	A1 Y: PROC	
SOL:G_S,20:<assertion>			SOL { A1_Y code RETURN;	
A1.Y: <assertion>			20	
: :			END A1_Y;	
: :				
A1.Y: <assertion>				
: :				

Figure 6.5c
Code Generation: Same Name and Common Qualifier

The first example, Figure 6.5a, corresponds to the case in which there is a one to one mapping between the set \mathcal{A} of assertions and the set \mathcal{S} of storage entries. This can occur when single assertions are found strongly connected by the cycles algorithm. The solution procedure is named as the assertion name corresponding to the first storage entry in the list (with "." substituted by "_" if a qualified name is used). Only a one level solution procedure is generated. The actual solution code shown in between brackets (SOL(\mathcal{A})) is explained in the following section.

Figure 6.5b presents a two level hierarchical system. the set of assertions under a single storage entry corresponding to AK will require a second level solution procedure. Finally Figure 6.5c shows an instance in which each storage entry contains a list of assertions. They also contain a common qualifier. In such cases the user is required to specify a "qualified list" of solution parameters, so that the processor can determine without ambiguity those corresponding to the global procedure from those associated with a single storage entry.

6.3.3 Code Generation

The following conventions are used in describing the Gauss-Seidel iterative solution generated code:

[TARGET] = {y1,y2,.....,yn}, the set of endogenous variables.

These are the target variables of a group of simultaneous assertions $\mathcal{A} = \{A1,A2,.....,An\}$.

[INIT] = {INIT.y1,INIT.y2,.....,INIT.yn}, is the set of initial values for endogenous variables. They are obtained from the corresponding list in D(i) for A(i).

[LAST] = {LAST.y1,LAST.y2,.....,LAST.yn}, a set of auxiliary variables for temporarily holding the last iteration values of endogenous variables.

[TEST] = {TEST.y1,TEST.y2,.....,TEST.yn}, is the set of convergence TEST values associated with each endogenous variable yi(i). It is obtained from corresponding list in data area D(i).

[\mathcal{A}] = {A1,A2,.....,An}, consist of the body of computational equations for the procedure

$$\text{eg. } y_1 = f_1(y_2, \dots, y_n, x_1, \dots, x_k, \theta, u_1)$$

.

.

$$y_n = f_n(y_1, \dots, y_{n-1}, x_1, \dots, x_k, \theta, u_n)$$

Whenever $NAME(A_i) = NAME(A_j)$ a CALL $NAME(A_i)$ is inserted instead of the assertion in the global procedure and a second level procedure $NAME(A_i)$ is generated with the same schema as described below. The second level procedure will contain all assertions in the same storage entry as A_i .

Conditional expressions are generated for assertions using the IF or FOR clause

$MAX_<proc-name>$ = The maximum number of iterations parameter for the procedure. The list of parameters is stored by the MODEL processor in the first data area of each storage entry S_i .

$STEP_<proc-name>$ = Optative step size parameter associated with the procedure. Used for generation of the improved SOR solution method. Also stored in the solution parameter list in each S_i .

Figure 6.6 shows the schematic implementation of the Gauss-Seidel generated algorithm. ITER is an auxiliary variable used as a counter for the number of iterations.


```

      <proc-name>: PROCEDURE
(1)      ITER = 0
(2)*      [TARGET] = [INIT]
(3)*      [LAST] = [TARGET]
(4)      ITER = ITER + 1
(5)      IF ( ITER > MAX_<proc-name> ) GOTO (10)
(6)*      [A]
(7)*      IF (ABS([TARGET] - [LAST]) > [TEST]) GOTO (3)
(8)      PRINT 'SOLUTION REQUIRED' ITER 'ITERATIONS'
(9)      RETURN
(10)     PRINT 'NO SOLUTION IN ' MAX_<proc-name> 'ITERATIONS'
(11)     RETURN
(12)     END <proc-name>

```

Figure 6.6

Code Generation Schema

* These steps contain "set expressions" which should be expanded for each element in the set. Operations between sets are done on a one-to-one basis. For instance step (2) is expanded as follows:

```

(2)  TARGET.Y1 = INIT.Y1
      TARGET.Y2 = INIT.Y2
      .
      .
      TARGET.YN = INIT.YN

```

The schema of Figure 6.6 can be modified to represent the SOR method by introducing the following code in order to change step size:

```

(1) --> (2)
(2.1) NT = 0
(3)  [TARGET] = [LAST]
      + STEP_<proc-name> * ([TARGET] - [LAST])
(3.1) [LAST] = [TARGET]
(4) --> (9)
/*---CHANGE STEP SIZE ---*/
(10)  STEP_<proc-name> = STEP_<proc-name> * 0.8
(10.1) ITER = 0
(10.2) NT = NT + 1
(10.3) IF( NT > NSTOP_<proc-name>) GOTO (11)
/*--- TRY TO CONTINUE ---*/
(10.4) GOTO (3)
(10.5) PRINT 'NO SOLUTION FOR' NT 'STEP CHANGES'
(11) --> (12)

```

where NSTOP_<proc-number> is a solution parameter indicating the maximum number of attempts at different step sizes and NT is an auxiliary variable used to accumulate the number of changes.

The corresponding PL/I code generated by SOLVEGS will be outputed to a PL/I procedure file PL1PROC.

6.4 Normalization

The motivation for, and effects of, different normalization rules were stated previously in section 6.2.2. A target or output set must be such that:

- 1 - Each assertion has exactly one target or endogenous variable,
- 2 - Each variable appears as the target of exactly one assertion.

Not always is possible to normalize an assertion. For instance the polynomial equation:

$$x^3 + x^2 - 2x = 0$$

contains the target variable "x" more than once, therefore an iterative method such as Newton-Raphson is required for solving it.

Algorithms for selection of a target set given an arbitrary system of unnormalized equations have been reported in the literature with applications in different disciplines. For instance Steward [STE 62] presents a method to obtain a first admissible output set and then shows how any other output set can be derived from relationships between a given set and its loops or cycles. Later Soylemez [SOY 71] extended the Steward and Rudd algorithm [RUD 64] into the SWS (Structural analysis With Substitution) algorithm for solution of equations found in chemical process design calculations. Soylemez's SWS

algorithm reduces systems of equations by heuristic selection of an output set, and then algebraic substitution of the set into the remaining equations. The algebraic substitution is carried out until an equation attains a given degree of complexity, based on its number of terms, type and degree of nonlinearities, etc.. This technique can be combined with the more efficient cycles processor in MODEL as follows:

- 1) Strongly-Connected Decomposition.
- 2) Heuristic Reduction of S-C Blocks.
 - a.- Selection of Output Set.
 - b.- Renormalization.
 - c.- Algebraic Substitution.
- 3) Generation of Solution Procedures.
 - Newton for unnormalized equations.
 - Gauss-Seidel for normalized nonlinear equations.
 - Matrix inversion for linear equations.

This modified version of the SWS algorithm, in the non-procedural context of MODEL can prove of advantage in certain engineering applications.

MODEL can be extended to include any of the existing algorithms, or allow the user to specify his target set before processing, or a combination of both. However any of these techniques will require the capability to "renormalize" an assertion previously stored in the AM for later code generation. A simple algorithm which take

advantage of the derivation tree structure of a stored assertion is described for such purposes in the following sections.

6.4.1 Normalization Algorithm

The detailed description of a "derivation tree" (DT) for an assertion in MODEL is given by Shastri [SHA 78]. For purposes of stating the basic algorithm a simplified tree of a "simple assertion" entry will be used. This entry is considered with the following format:

```
-----
| LHS-PTR | RHS-PTR |
-----
```

where

LHS-PTR is a pointer to the DT of the arithmetic expression in the left-hand-side of the assertion, and

RHS-PTR is a pointer to the DT of the arithmetic expression in the right-hand side of the assertion.

Symbolically an assertion such as

$$(A - B) + C * D = X$$

will be represented as depicted in Figure 6.7

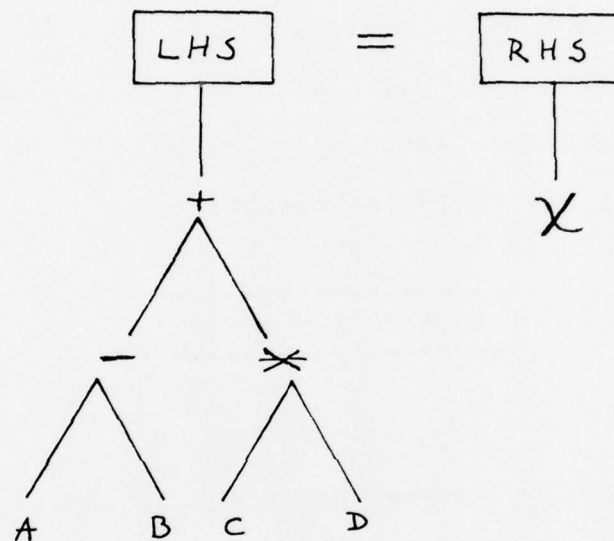


Figure 6.7

Symbolic DT for an Assertion

The algorithm is given as inputs a pointer to the DT of the assertion and the target variable for which it should be normalized for. The algorithm transform the given DT into a normalized DT with a single terminal node at the LHS; the target or dependent variable. For instance if "A" is

selected as the target variable in the assertion whose DT is depicted in Figure 6.7, the structure is transformed into the DT shown in Figure 6.8, representing the normalized assertion

$$A = X - C * D + B$$

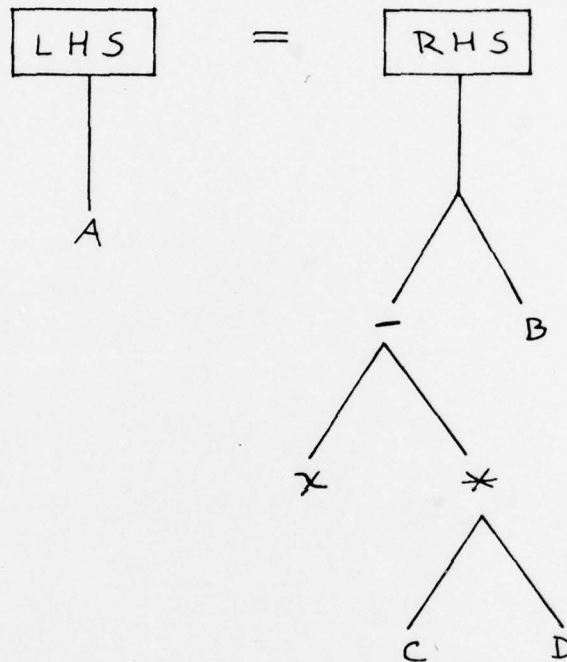


Figure 6.8

Normalized DT Corresponding to Assertion in Figure 6.7

Basically the algorithm requires the capability of taking inverses of functions and operators in the language, checking if the operators are commutative or not, a Depth-First-Search (DFS) procedure to find the target element in the tree and redefining pointers of subtrees

elements or leaves.

Each node of the tree pointed by LHS-PTR or RHS-PTR is assumed with the following format:

```
-----
| OP | FIRST-PTR | SECOND-PTR |
-----
```

where OP contains the entry (operator or terminal variable),
FIRST-PTR is a pointer to the next node in the tree,
 or NULL if it is a terminal node, and
SECOND-PTR is a pointer to the second node for binary
 operators or functions, and it is NULL for unary
 operators and terminal nodes.

The actual structure used by MODEL is much more complex, but the algorithm can be stated without loss of generality using this simplification.

The following functions and internal procedures are used in describing algorithm NORMAL:

INV(OP): Search table of inverse operators and returns the
 inverse of OP.

SEARCH(P,T,S): Performs a DFS in the tree pointed by P,
 until terminal variable T, the target, is
 found. Pointer S is set to the subtree out of
 the root of the P tree that contains T, or to
 NULL if not found.

CHANGE(P1,P2): Exchange pointers P1 and P2, eq.

P \leftarrow P1

P1 \leftarrow P2

P2 \leftarrow P

MOVE(S,P,R,F): Node pointed by R becomes the root of tree pointed by P. F is a logical variable. If F is set to true ('1'B), then the pointers of R are set to the subtree pointed by S, and to the old root element pointed by P respectively. If false ('0'B), the first pointer of R is set to the old root element pointed by P, and the second pointer to the subtree pointed by S.

COMM(OP): Returns a bit '1'B if OP is a commutative operator, otherwise returns a '0'B. (e.g., $A/B \neq B/A$, therefore $\text{COMM}('/') = '0'B$).

A recursive algorithm NORMAL can now be described with the following steps:

Step 1: Search for the target variable T in LHS. If not found repeat in RHS. If T is in RHS switch pointers of LHS with RHS, else the assertion does not contain the target T and there must be an error.

Step 2: If the root pointed by LHS is a terminal node, then STOP; the assertion is normalized. Else if target T is in branch pointed by FIRST-PTR proceed to Step 3. If target T is in branch pointed by the SECOND-PTR

check if the root operator is commutative (COMM(OP)).
If so, proceed to Step 3, else set logical variable
F='1'B and go to Step 4.

Step 3: Set F='0'B and take inverse of root element
(INV(OP)).

Step 4: Move root and subtree not containing T to the RHS.
Old RHS tree becomes subtree pointed by FIRST-PTR of
new RHS root if F='1'B, or by SECOND-PTR if F='0'B.

Step 5: Pop up LHS, that is LHS should now point to subtree
containing T.

Step 6: Repeat Step 1 to Step 3.

Figure 6.9 shows the necessary PL/I code for
implementing algorithm NORMAL while Figure 6.10 gives an
example of its application to an unnormalized equation.

```

NORMAL: PROCEDURE (LHS-PTR, RHS-PTR, T) RECURSIVE
  CALL SEARCH (LHS-PTR, T, S)
  IF S ~ NULL THEN GOTO NORM
  CALL SEARCH (RHS-PTR, T, S)
  IF S = NULL THEN STOP /* ERROR */
  CALL CHANGE (RHS-PTR, LHS-PTR)
NORM: IF (LHS-PTR->FIRST-PTR = NULL
      & LHS-PTR->SECOND-PTR = NULL)
      THEN RETURN /*ASSERTION NORMALIZED */
  IF (S=LHS-PTR->FIRST-PTR | COMM(LHS-PTR->OP) THEN
  DO
    F = '0'B
    LHS-PTR->OP = INV(LHS-PTR->OP)
  END
  ELSE F = '1'B
  IF (S = LHS-PTR->FIRST-PTR) THEN
    CALL MOVE (LHS-PTR->SECOND-PTR, RHS-PTR, LHS-PTR, F)
    ELSE CALL MOVE (LHS-PTR->FIRST-PTR, RHS-PTR, LHS-PTR, F)
  /*-- POP UP LHS --*/
  LHS-PTR = S
  CALL NORMAL (LHS-PTR, RHS-PTR, T)
  RETURN
END NORMAL

```

Figure 6.9

PL/I Code for Normalization Algorithm

TARGET: A;

$$(A * B) + \text{LOG}(C + D) - (E * F) = X;$$

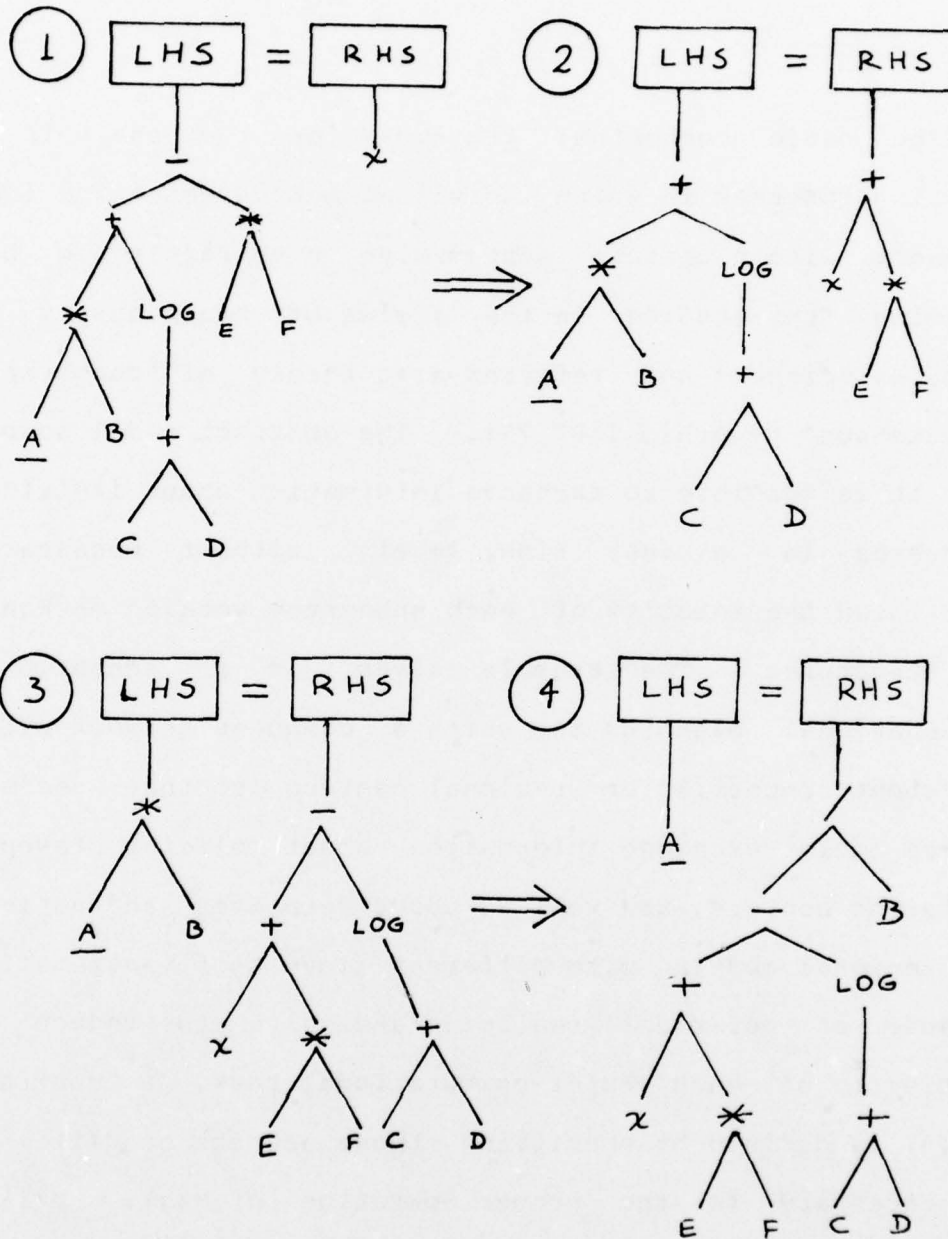


Figure 6.10

Example of Normalization for a DT

CHAPTER 7

Model Linkage and Cooperative Computation

7.1 Introduction

The basic conceptual framework for the analysis of certain processes in which individual subsystems enjoy local autonomy within certain system-wide constraints has been evolving from studies in the fields of brain theory and computer sciences and referred as a theory of "cooperative computation" by Arbib [ARB 77]. The abstract model assumes that it is possible to exchange information about individual processes in a very high level, without necessarily describing the totality of each subsystem working mechanism or structure. The example given is a scenario of international planning in which a computer network links different countries or regional centers so that decision makers could exchange information about policies taken at different centers, as well as about databases and national or regional models with different levels of aggregation. Because of political realities and also to reduce the complexity of each center computational task, a "contract" $C(i,j)$ is defined by specifying those aspects of $M(i)$ which are necessary for the proper operation of $M(i)$. $M(i)$ in turn could be substituted by $\hat{M}(i)$ if

- i) $\hat{M}(i)$ satisfies each $C(i,j)$ and
- ii) satisfaction of each $C(i,i)$ suffices for proper operation of $\hat{M}(i)$.

The basic model presented by Arbib requires a high level descriptive language as well as mechanisms which will allow this interaction through appropriate contracts. With the advent of international computer networks (eq. GE MARK III, ARPANET) and distributed computation technology, the hypothetical international planning scenario could be closer to reality, yet numerous obstacles ranging from the spectrum of different political realities to more crude problems of modeling methodology and data standardization will have to be overcome to achieve that objective. However in this chapter a description of the LINK world trade model [BALL 73, KLE 76a] will illustrate a working example of cooperative computation which can be closely described by an even more refined abstract model. A reduced form of the LINK system will then be specified in MODEL to demonstrate how this very high level non-procedural language can be used as a basic tool from which a knowledge database (the "expert system" in Arbib's terms) for different applications can evolve in this context.

7.2 The LINK World Trade Model

The LINK system is a large scale econometric system consisting of separate models for: i) 20 countries, ii) regional models for four developing areas and iii) a residual group of "rest-of-the-world" countries*. The objective of the integrated system is to produce better predictions of world trade by imposing consistency in the interacting variables through a trade model and the link algorithm. The configuration is also used to study the transmission of international economic fluctuations. A decentralized schema was chosen for implementation since the philosophy of the project is that individual country members are responsible for their models and in the best position to judge local economic developments as well as to recommend particular structures and specifications.

Each center in turn benefits from the configuration by allowing their full scale national models to be imbedded into a consistent scenario, yielding a better overall performance of its aggregates.

The following description of the algorithm presently

* The dynamics of the project assures that new countries or regional models are regularly being added or updated. Some versions have used reduced form equations for some 12 other developed countries. Also the current system includes linkages between some 40 commodity models and the country or regional models. To avoid unnecessary complications, these will not be included in the general description. The structural characteristics of each of the models is documented in [WAE 76].

used in solving the LINK system will help to illustrate the nature of cooperative computation involved:

Step 1: Each center makes best estimates of initial conditions and exogenous variables and submits a "control solution" to the LINK center, which acts as a coordinator.

Step 2: The coordinator makes final adjustments (normally this phase involves some translation process from the center data and code representation to that one used by the coordinator), and loads the solution into the coordinator database.

Step 3: Import and export prices which are determined by each national model are converted to merchandise only, FOB valuation, in current U.S. dollars. This requires exchange rates, FOB/CIF ratios and determination of the "goods" portion of goods and services, depending on the unit used by each center national model.

Step 4: Exports are determined in the trade model from the relationship

$$XS(t) = A * MS(t)$$

where

$XS(t)$ = column vector of exports in period t , measured in constant U.S. dollars.

A = trade shares matrix ($A(i,j) = X(i,j)/X(.,j)$)

$X(i,j)$ = merchandise shipments from country i to country j .

$X(.,i) = \sum_i X(i,i)$ = imports of country i

$MS(t)$ = column vector of imports in period t ,
measured in constant U.S. dollars.

By construction the columns of A sum to unity

$$\sum_i A(i,i) = 1$$

This assures that the basic identity

$$\sum_i XS(i) = \sum_i MS(i) = TWS$$

TWS = world exports FOB = world imports FOB

Step 5: Import prices are evaluated as column weighted averages of other countries export prices.

Since the identity of step 4 holds also in current prices:

$$(PXS(t))'XS(t) = (PMS(t))'MS(t)$$

where

$PXS(t)$ = index of dollar denominated export prices
in period t .

$PMS(t)$ = index of dollar denominated import prices
in period t .

or

$$(PXS(t))'AMS(t) = (PMS(t))'MS(t)$$

from which

$$PMS(i,t) = \sum_i PXS(i,t)A(i,i) \quad i=1,2,\dots,n$$

Step 6 : Other significant interacting variables which are direct linkages between models or groups are exchanged after appropriate transformations and unit conversion. These variables are referred as "off-diagonal" variables since they are exogenous to

some models but endogenously determined by others.

Step 7 : National or regional models are successively re-solved with new input values for exports and import prices, as well as off-diagonal variables, as determined in steps 4 through 6.

Step 8: Steps 3 through 7 are repeated with new solution values. At each iteration world trade is computed as the sum of world imports (FOB) in current dollars and convergence is assumed when $TWS(t)$ does not change from iteration to iteration more than a preassigned convergence test value. At this stage it should be checked that all country specific variables are also convergent after reiteration. The consistent final solution is stored on the system database and used as lagged input values for the next period solution.

Notes: 1) Since the trade shares matrix "A" does not remain stable over time, the identity in step 4 holds only at period "t" and errors are generated when used outside the base period ($XS(t) = A(0)MS(t) + \text{error}(t)$). Adjustment factors are introduced to account for export shifts as functions of relative prices:

$$XS(i) = \alpha(i)PXS(i) + \beta(i) \left(\sum_j A(i,j)^{70} \right) MS(i) + \gamma(i)PCOMS(i) + \delta(i)t$$

where

$\sum_j A(i,j)^{70}$ = ith row of A matrix for 1970

$PCOMS(i)$ = index of dollar denominated export prices competitive with i's exports.

While the trade model equations are computed for each commodity group, the above equation is presently used only for manufactures (SITC 5-9). This approach, when properly restricted, has certain analogies with the linear expenditure system (LES) and it is one of several schemes

presently being experimented to account for shifts in "A" over time [CAN 77].

When $XS(i)$ are computed from the previous equations, it is found that the distribution of exports over countries through

$$\sum_j A(i,j)^{70} MS(j)$$

will not add to computed totals $XS(i)$, therefore a standard RAS adjustment to the estimated trade matrix " \hat{A} " is done at each iteration to assure complete accounting balance, with row sums adding to the estimated export totals for each country.

2) The trade model equations were set out in terms of constant dollars trade matrices. The model could also be stated in terms of current dollars trade matrices by price conversion through weighted harmonic means rather than arithmetic means.

It is interesting at this stage to draw the analogy between the LINK system and the basic model of cooperative computation introduced in the previous section. The equivalent to direct bilateral contracts ($C(i,j)$) are reduced only to the "off-diagonal" relationships. Complete direct linkages by introducing equations explaining each bilateral trade flow have been found not practicable at present, since information about those flows is difficult to come by [RHO 73]. Instead a trade shares model is used in an iterative schema, which can be looked upon as an "intermediate" level process "T", which receives the necessary interacting variables from each country " $L(i,T)$ " (i.e., PXS , MS) and produces a consistent new set of interacting variables " $L(T,i)$ " (i.e., PMS , XS) for the working of each national model.

Direct bilateral contracts are then substituted by

proper overall agreements with an intermediate coordinator. The role of the reduced process $\hat{N}(i)$ will become clear while explaining a reduced form of the LINK system in next section.

Figure 7.1 shows an schematic diagram for solving the LINK system in the context of MODEL. Individual countries are specified in the top layer with their corresponding source and target files ($N(i) = \{S(i), T(i), A(i)\}$, $i=1$ to n and $A(i)$ is a set of functions mapping the source set $S(i)$ into the target set $T(i)$ i.e., $A(i) = \{S(i) \xrightarrow{f} T(i)\}$, the set of assertions). These national models are depicted as stand alone specifications or "modules" to keep with the decentralized philosophy of the project, allowing "local" or isolated solution experiments. Off-diagonal relationships ($C(i,j)$) as well as other interacting variables for the linkage mechanism ($L(i,j)$) are also indicated by the arrows in the modules. Detailed description of the linkage mechanism is illustrated in the bottom part of the diagram. The actual linkage variables used by the LINK system are explicitly shown here. The country models will usually have export volume ($X(i,k)$) and import prices ($P^M(i,k)$) as exogenous or source variables, whereas assertions will be provided to compute endogenous or target variables for import volume ($M(i,k)$) and export prices ($PX(i,k)$). The subscript "i" runs over the countries and "k" identifies each SITC commodity group. The "interface" modules describe transformations from the national model units into a single

and convenient numeraire (U.S. dollars) using the countries exchange rate ($EX(i)$). Other transformations to cope with possible CIF/FOB valuations, standard disaggregation into SITC categories and common data periodicity may also be included in this module together with appropriate "inverse" transformations from variables obtained from the trade model ($XS(i,k)$ and $PM(i,k)$) back into national units.

Finally the trade model is shown with the description of the linkage algorithm and references to the stored specification of each separate module in the system, for later generation of a global solution program by the MODEL processor.

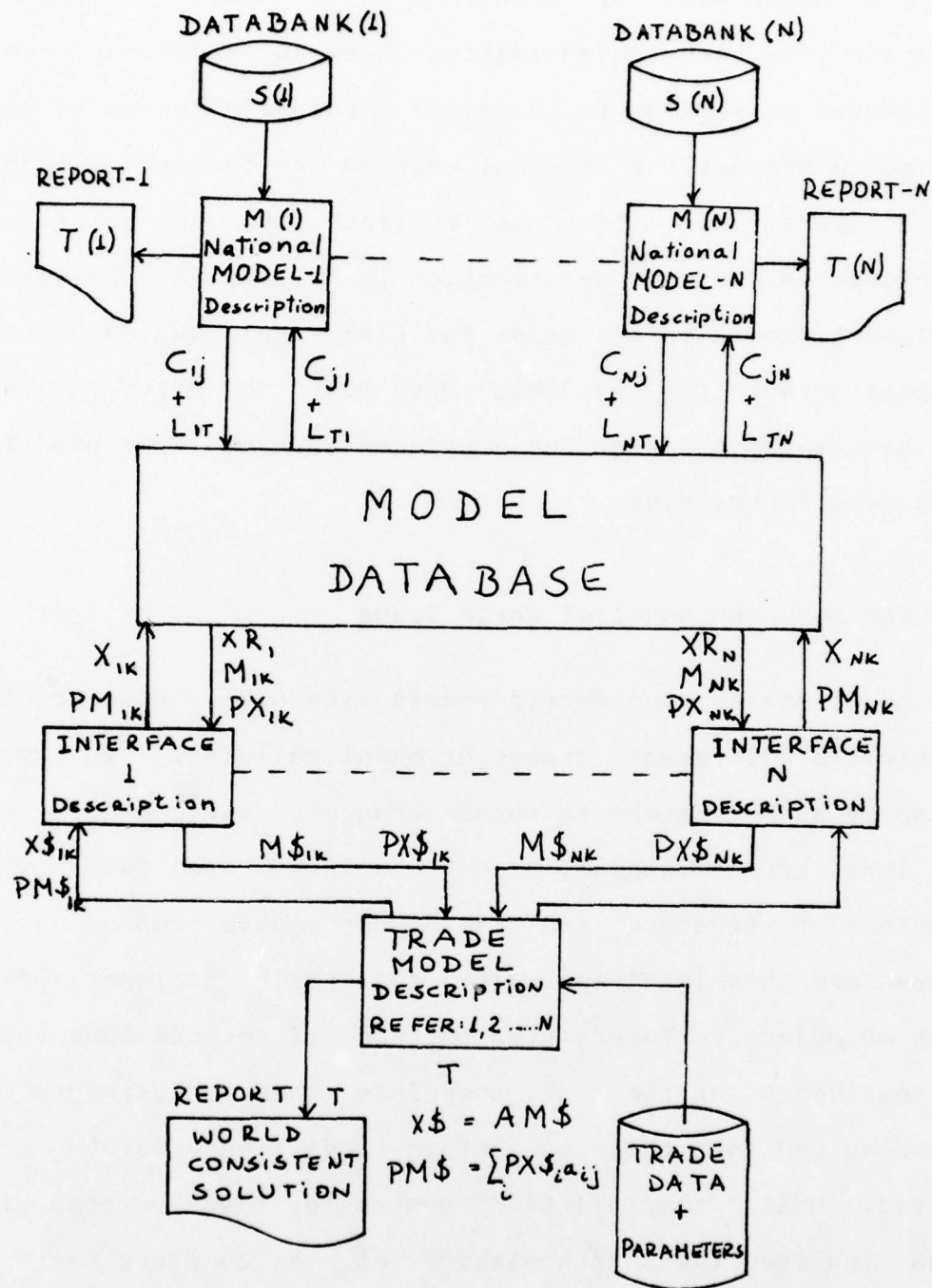


Figure 7.1

Schematic Diagram for LINK System

The advantages of using a high level descriptive language for both communicating between and about these structures as well as to generate a solution system without having to set out the internal working intricacies of such a large system can best be realized by comparing the complexities of such specification in MODEL with an existing implementation used to solve the LINK model and documented by this author in [GAN 76b]. With that objective in mind, the next section describes a reduced form of LINK used for long term forecasting.

7.3 The Long Run Model of World Trade

Large scale econometric models like LINK, pose special problems at different stages of model building. In trying to apply such systems to obtain economic variables for use in long term planning, model builders are faced with problems of structure and size. Most models composing the system are highly disaggregated full scale national models with an objective forecasting horizon of no more than three or four years at the most, therefore major adjustments may be required for them to obtain predictions beyond this period. Using the full LINK system for this purpose will also involve the extrapolation of a complete set of exogenous assumptions, a task of enormous dimensions to be practicable. To avoid these problems, project LINK developed a long term world trade model denoted as MAXILONG [KLE 76b], which is conceptually equivalent to the full

scale LINK system (MAXILINK), but simple and manageable in size, using only a reduced set of exogenous inputs which can be easily obtained.

Before describing the actual set-up for MAXILONG, it is illustrative to refer back to the model of cooperative computation and its reduced form process element $\hat{M}(i)$. The conditions for $\hat{M}(i)$ to substitute $M(i)$ were set to be the satisfaction by $\hat{M}(i)$ of each individual contract $C(i,j)$, and that each individual contract in turns suffices for proper operation of $\hat{M}(i)$. In the context of LINK, the subsystems are connected via interacting endogenous and exogenous variables with the trade model $(L(i,i))$ and direct off-diagonal relationships $(C(i,i))$. Therefore each reduced model $\hat{M}(i)$ could consist of only the necessary equations to determine the interactive endogenous variables $(Y_i^I \in L(i,T))$ as reaction functions of other interactive exogenous variables $(X_i^I \in L(T,i))$ plus the trade model to meet the overall system constraints. For simplicity the direct off-diagonal relationships will not be considered. The general form of an equation i of $\hat{M}(i)$ is of the form:

$$Z(i,i) = F_{ii}(Z(j), X(i), A(i,i))$$

where

$Z(i,i)$ = the price of exports (or volume of imports) of the i th SITC commodity group ($i=01, 20, 3, 5-9$).

$Z(j)$ = the vector of all endogenous variables in $M(j)$.

$X(j)$ = the vector of all exogenous variables in $M(j)$

$A(i,j)$ = vector of parameters.

Further, MAXILONG uses the same functional form for all equations over $\hat{M}(j)$ for all countries j , therefore

$$Z(i,j) = F(Z(i), X(i), A(i,j))$$

A minimal set of exogenous variables was chosen as consisting of the price of imports (PM) and quantity of exports (X), which are the exogenous interactive variables to each country model. The equations are specified as follows:

$$Z(k,j,t) = A(1) + A(2)*t + \sum_{l=0}^N B(l)*PM(k,j,t-l) + \sum_{l=0}^N C(l)*(VX(k,j,t-l)/PX(k,j,t-l))$$

Z = export price (PX) or import volume (M)

The indices $k, j, t-1$ represent the k th SITC commodity group for $\hat{M}(j)$ at time $t-1$ respectively. The parameters B 's and C 's can be obtained from the structural models by computing appropriate multipliers (refer to [KLE 76b] for detailed account of the calculations followed).

These reduced form models $\hat{M}(i)$ will be used in the next section together with a trade shares algorithm to describe MAXILONG in MODEL.

7.4 MODEL Description of MAXILONG

The specification for solving MAXILONG that follows will use a set-up such as the one described for solving the complete LINK system in Figure 7.1, that is each country module must be able to run both as stand-alone model, for individual adjustments, and imbedded in the overall world trade model for consistent solution.

This subsection presents first an overview of a single country reduced economic model followed by an integrated multi-country model. This is a simplified description of MAXILONG, omitting some details that were not considered essential, and which will be referred as MINILINK.

Figure 7.2 illustrates an individual country reduced model. The rectangular box in the center represents the desired program for simulating the model of country "A". Only four variables are present: the price indices of imports and exports (P^M and P^X), the value of imports (V^M) and the quantity of exports (X), where $X = V^X/P^X$ and V^X represents the value of exports. P^M , and X are the exogenous variables. The data about these variables comes from a data base named PANK. The reduced model of country "A" consists of only two equations with the two endogenous variables V^M , P^X . The starting year (referred to as LINK_YR) and end year (END_YR) of simulation, as well as the values of the coefficients (COEFF) to be used in the structural equations (estimated previously) come from an

INPUT data base. The results of the simulation formatted in a report are placed in the SOLUTION data base.

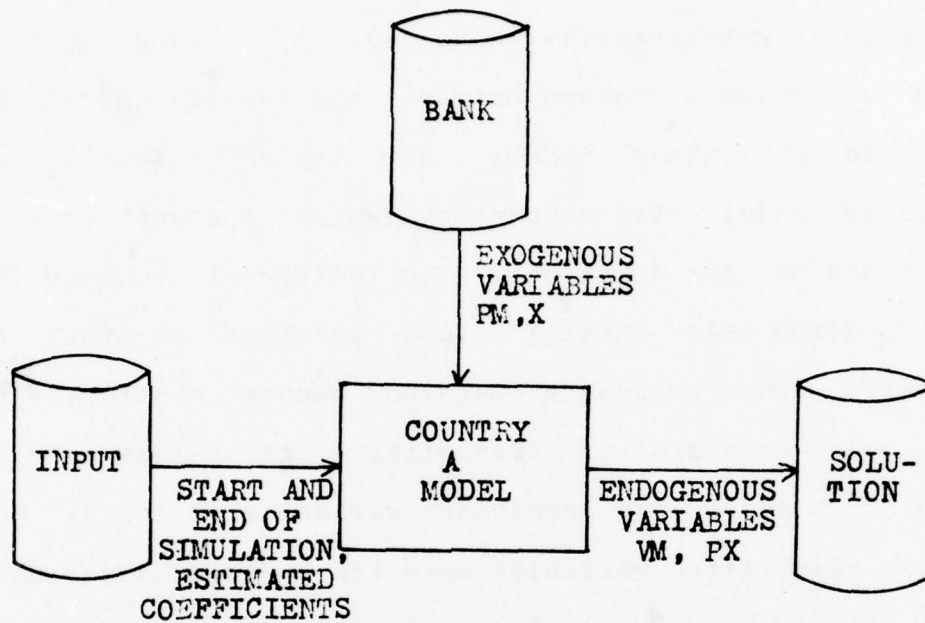


Figure 7.2

Block Diagram for Country "A"

The integration of the models of a number of countries in MINILINK is shown in Figure 7.3. The integrated model consist of a) the individual models of the countries in a form which is substantially unchanged, b) a trade model and c) the variables exchanged between the country models and the trade or linkage model. All the variables in the integrated model are endogenous (which account for the consistency of the solution). The individual country data banks provide only initial values and lags to start the simulation. The "contracts" between country models and the trade model consists of transmitting the values of the variables PX and VM and receiving variables PM and X. Note that the transmitted variables were the endogenous variables in the single country model, while the received variables were the exogenous variables.

MINILINK uses the TPADAT data base which contain information on merchandise trade between partner countries for the year 1970 (the trade shares matrices). The assumption here is that the trade shares will be constant throughout the simulation period. A70 therefore can be considered as a constant parameter or as a new exogenous variable.

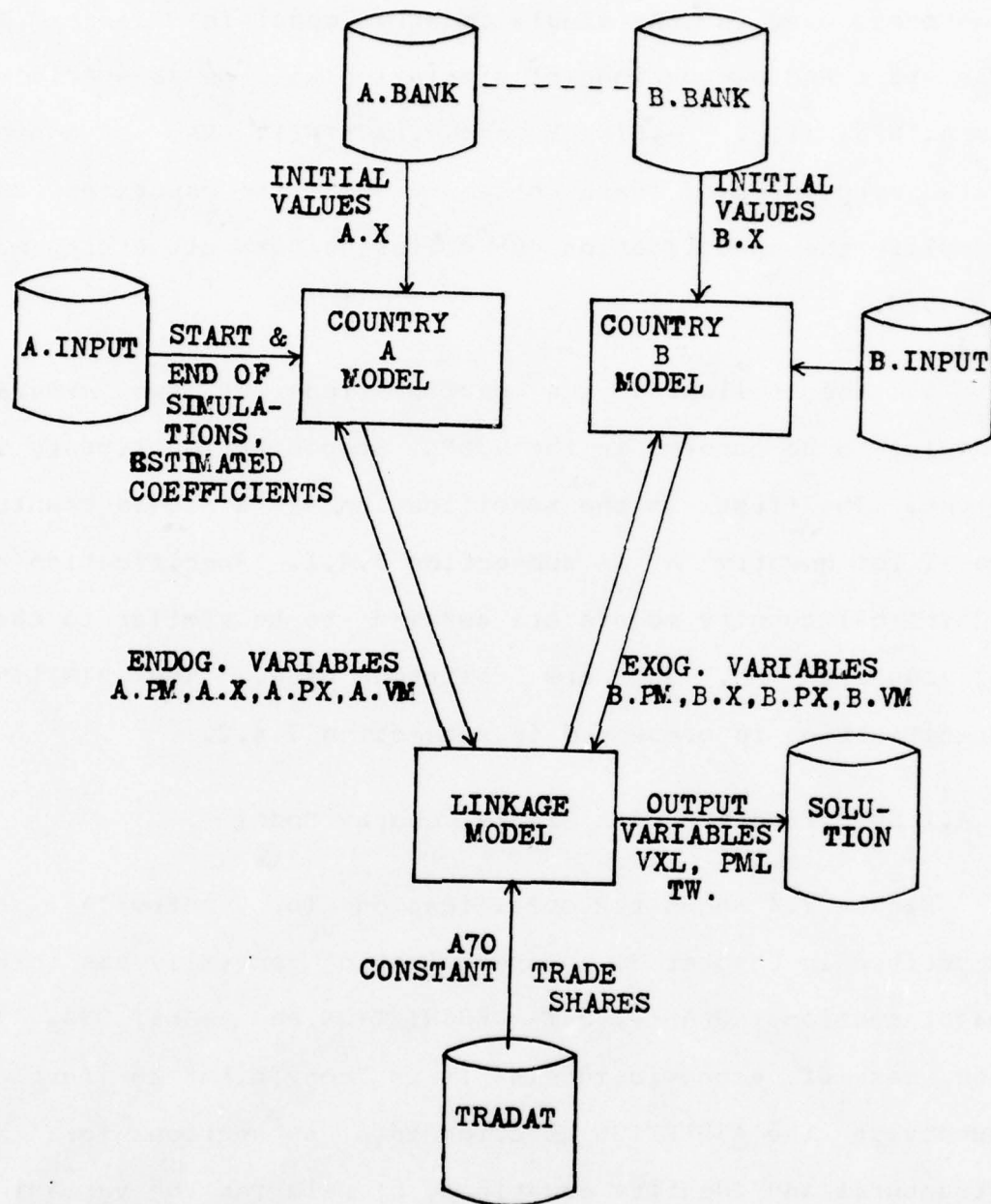


Figure 7.3

Block Diagram of Linked Trade Model MINILINK

The BANK, INPUT and SOLUTION data bases are similar to the ones used in the single country model in Figure 7.2. The start and end periods of simulation will be as specified in A.INPUT (i.e. A.LINK_YR and A.LINK_NP). Use of common data structures of data bases by all the countries may simplify the specification but different data structures may be used.

In the following the specifications of two program modules to be stored in the MODEL Specification Library is given. The first is the specification of a single country model for country "A" in subsection 7.4.1. Specification of additional country models are assumed to be similar to that of country "A", and are omitted here. The MINILINK specification is presented in subsection 7.4.2.

7.4.1 Specification of a Single Country Model

Figure 7.4 shows the specification for Country "A". As described in Chapter 3, a specification generally has three major sections: HEADER, DATA_DESCRIPTION and ASSERTIONS. In the case of economic models it is convenient to further subdivide the ASSERTION section into subsections for; a) structural and identity equations, b) relating the variables in the model to data in the data bases. The ASSERTION section in Figure 7.4 is subdivided into subsections of A_STRUC_EONS (structural equations), SOLUTION_ASSERTIONS and AUXILIARY ASSERTIONS.

The HEADER provides the module name - A, and the names of the source files: BANK and INPUT, and the target file: SOLUTION. This corresponds to the block diagram in Figure 7.2 for the model of Country "A".

The DATA_DESCRIPTION section contains a) descriptions of the data bases (BANK, INPUT and SOLUTION) b) subscripts and c) interim variables. BANK is a data base that contains economic statistics of the country. It consists of records (referred to as TIM_SER), each corresponding to a time series for one economic variable. The record contains a number (NUMBER) and a label (LABEL) identifying the variable, followed by the data of the time series (DATA).

In line 10 starts the description of the INPUT file. As shown, it contains one record named CONTROLS which contains LINK_YR, the start year for the simulation, LINK_NP, the number of periods to be simulated, and finally COEFF, a list of the values of the previously estimated coefficients for the structural equations.

The description of SOLUTION file starts in line 15. This is a report and each record in this file corresponds to a line in the report. For each period, the report will consist of a group of records named PERIOD. PERIOD contains a HEADER record (line) naming the various column headings, a line for each COUNTRY with the price P and value V, concluding with a TOTAL record which gives TW, the total trade for the period.

/* SPECIFICATION OF MODEL OF COUNTRY "A" */

SEC HEADER

1 MODULE: A;
2 SOURCE FILES: BANK, INPUT;
3 TARGET FILES: SOLUTION;

SEC DATA_DESCRIPTION

4 DISK IS MEDIA(UNIT=3330);
5 BANK IS FILE(DISK,ORG=SEQ,KEY=NUMBER,FORMAT=FIXED);
6 TIM_SER IS RECORD(BANK,(5,4));
7 NUMBER IS FIELD(TIM_SER,NUM(4));
8 LABEL IS FIELD(TIM_SER,CHAR(8));
9 DATA IS FIELD(TIM_SER,DEC(14,5),(20));
10 INPUT IS FILE(DISK,ORG=SEQ,FORMAT=FIXED);
11 CONTROLS IS RECORD(INPUT,(1));
12 LINK_YR IS FIELD(CONTROLS,NUM(4));
13 LINK_NP IS FIELD(CONTROLS,NUM(4));
14 COEFF IS FIELD(CONTROLS,NUM(4),(12));

Figure 7.4

Specification of Country "A" Reduced Economic Model

```
15 SOLUTION IS REPORT(DISK,ORG=SEQ,FORMAT=VARIABLE);
16 PERIOD IS GROUP(SOLUTION,(LINK_YR:END_YR));
17   HEADER IS RECORD(PERIOD,(1));
18     TIME IS FIELD(HEADER,CHAR(10));
19     NAME IS FIELD(HEADER,CHAR(20));
20     PRICE IS FIELD(HEADER,CHAR(20));
21     VALUE IS FIELD(HEADER,CHAR(20));
22 COUNTRY IS RECORD(PERIOD,(NO_CNTRY));
23   FILLER IS FIELD(COUNTRY,CHAR(10));
24   CNTRY_NAME IS FIELD(COUNTRY,CHAR(20));
25   P IS FIELD(COUNTRY,PIC'(14)Z.V(5)9');
26   V IS FIELD(COUNTRY,PIC'(14)Z.V(5)9');
27 TOTAL IS RECORD(PERIOD,(1));
28   FILLER IS FIELD(TOTAL,CHAR(50));
29   TW IS FIELD(TOTAL,PIC'(14)Z.V(5)9');
```

Figure 7.4 (continued)

```
/* SUBSCRIPT DESCRIPTIONS */  
30 T IS SUBSCRIPT(PERIOD, LINK_YR, END_YR, 1); /*SOLUTION  
TIME*/  
31 I IS SUBSCRIPT(COUNTRY, 1, NO_CNTRY, 1); /*COUNTRY*/  
32 K IS SUBSCRIPT(TIM_SER, 1, 4, 1, 2); /*SITC CATEGORY*/  
33 DT IS SUBSCRIPT(DATA, 1, 20, 1); /*DATA TIME*/  
  
/* INTERIM VARIABLES */  
34 PM, X ARE INTERIM((20, 4), DEC(14, 5)); /*EXOGENOUS  
VARIABLES*/  
35 PX, VM ARE INTERIM((LINK_YR:END_YR, 4), DEC(14, 5)); /*ENDO  
G  
VARIABLES*/  
  
/* INTERIM AUXILIARY VARIABLES USED TO SPECIFY  
SOLUTION FILE DATA */
```

Figure 7.4 (continued)


```
36 X09 IS INTERIM((LINK_YR:END_YR),DEC(14,5));
```

```
SEC ASSERTIONS
```

```
/* ASSERTIONS FOR RELATING A.BANK DATA TO EXOGENOUS  
COUNTRY "A" VARIABLES */
```

```
37 A.PM(DT,K) = A.DATA(1,k,DT);
```

```
38 A.X(DT,K) = A.DATA(5,K,DT);
```

```
SEC A_STRUC_EONS
```

```
/*REDUCED COUNTRY "A" MODEL STRUCTURAL EQUATIONS*/
```

```
39 A.VM(T,K) = A.COEFF(1) + A.COEFF(2)*T  
+ A.COEFF(3)*A.PM(T,K)+A.COEFF(4)*A.PM(T-1,K)  
+ A.COEFF(5)*A.X(T,K) + A.COEFF(6)*A.X(T-1,K);
```

```
40 A.PX(T,K) = A.COEFF(7) + A.COEFF(8)*T  
+ A.COEFF(9)*A.PM(T,K)+A.COEFF(10)*A.PM(T-1,K)  
+ A.COEFF(11)*A.X(T,K) + A.COEFF(12)*A.X(T-1,K);
```

Figure 7.4 (continued)

SEC SOLUTION_ASSERTIONS

/* ASSERTIONS FOR DATA IN SOLUTION FILE */

```
41 TIME(T) = T;
42 NAME(T) = 'NAME';
43 PRICE(T) = 'PRICE OF EXPORTS';
44 VALUE(T) = 'VALUE OF IMPORTS';
45 CNTRY_NAME(T,I) = 'A';
46 X09(T) = SUM(X(T,K),K);
47 P(T) = SUM(PX(T,K)*X(T,K)/X09(T),K);
48 V(T) = SUM(VM(T,K),K);
49 TW(T) = V(T);

/* AUXILIARY ASSERTIONS */
50 END_YR = LINK_YR + LINK_NP;
51 NO_CNTRY + 1;
```

Figure 7.4 (continued)

The remaining parts to be specified in the DATA_DESCRIPTION section are the subscripts and interim variables. The subscripts are described in lines 30 through 33 in figure 7.4. Subscripts are free variables intended to express indices of elements in arrays. The complete specification for a subscript consists of five parameters enclosed in parenthesis: 1) the name of the array where the subscript may be used to denote an index of an element, 2) the lower bound that a subscript may attain, 3) the corresponding upper bound, 4) the step size used in incrementing (or decrementing) the subscript, 5) the dimension number, when the array is of dimension greater than one. For instance the mnemonic T (for time) in line 30 will be used as a subscript of the array PERIOD in the SOLUTION file. The lower bound is LINK_YR, the starting year for the simulation. The upper bound is END_YR, the last year for the simulation. Similarly the other subscripts are defined: I to express an index for the country, K for each SITC commodity group and DT for a year in the time series (DATA) entries. In the example there are 20 periods for DATA and 4 commodity groups. Note that when a subscript is used to denote indices of several arrays, it needs to be described only once. The MODEL system will automatically propagate it to the other arrays based on use of subscripts in assertions.

Interim variables are divided into three groups: Exogenous, Endogenous and Auxiliary variables. As noted

before PM and X are the exogenous variables representing price index of imports and quantity of exports, respectively. The endogenous variables are PX and VM, representing the price index of exports and value of imports, respectively. Finally to obtain the total value of exports V reported in SOLUTION it is necessary to sum X over all SITC commodity groups. This sum is denoted by X09.

The next section in Figure 7.4 is that of ASSERTIONS. The first group consist of the assertions in lines 37 and 38 that relate the exogenous variables to the data provided in BANK. The second group of assertions correspond to the structural equations of the model for country "A". These are given in lines 39 and 40 in Figure 7.4 normalized for the endogenous variables. Finally the assertions that relate the data in the SOLUTION file to the endogenous variables are given in lines 41 through 49. The first five lines, 41 through 45 specify the strings of characters to be used to label the columns of the SOLUTION report.

The SUM function used in lines 46 to 48 is similar to the \sum symbol used in mathematics with the parameters (<expression>,<index>). The expression given as the first parameter is to be summed over all the values of the subscripts given in the second parameter. As noted in 46, X09(T) is a sum of the values of exports X summarized over all the values of the SITC category K. Line 47 specifies the weighted average of the price of exports PX over all the

categories. Note that in line 49 the variable TW represents total trade of the countries, and is set to V in this case as only one country is included in the model. Finally line 50 shows that END YR, which is a variable used to denote the last year of the simulation, is equal to the starting year (LINK YR) plus the number of periods (LINK NP).

7.4.2 Specification of the Trade Model: MINILINK

Similar to the previously described specification for Country "A", the MINILINK specification in Figure 7.5 consists of sections: HEADER, DATA DESCRIPTION and ASSERTIONS.

The HEADER section specifies the name of the module, the source data and target data. These correspond to the block diagram in Figure 7.3. In addition there is a REFER statement in line 4. This is to include in the specification of MINILINK data and assertions previously specified for the modules of individual countries. The sections, statements and their propagates that are referred to are copied from previously specified modules and included automatically in the module.

The specification of MINILINK in Figure 7.5 shows integration of only two country modules, "A" and "B". Additional country modules could have been included as well through referencing of their data bases and assertions in a similar manner in the REFER statement. For reasons of

brevity additional country models have been omitted. The DATA_DESCRIPTION section needs not include the files referenced in the REFER statement in the header. The only new data base added in MINILINK is TRADAT as shown in lines 5 to 7. TRADAT consists of a single record named SHARE which contains the matrix of trade shares from country to country, and for each SITC commodity group. This is therefore a three dimensional array.

The DATA_DESCRIPTION section includes in lines 8 through 12 descriptions of the subscripts, and in lines 13 through 17 the descriptions of the interim variables.

The ASSERTIONS section consists of several groups of statements. The first group specifies the simulation period (the number of elements in the solution array) and the total number of countries (NO_CNTRY). As shown the starting year of the simulation and the number of periods to simulate will be accepted from A.LINK_YR and A.LINK_NP, the input to the country "A" model.

The two next groups of statements specify the relations of the linkage variables to the trade model. These are the variables which are communicated between components of MINILINK. These relations are stated in lines 21 through 28. Initial lagged values for the quantity of exports are specified in lines 29 through 32. Finally the assertions for the endogenous linkage variables are given in lines 33 through 35. The ASSERTIONS section concludes with the

assertions specifying the relationships of the variables in the SOLUTION file to the variables in the linkage model.

In summary, the specification state assertions which relate variables in various submodels, one to the others, and variables in source and target data bases to the variables of the models. In addition the linkage equations are also provided. Notice that each country model equations and their respective data bases are incorporated without change by referring to them in the REFER statement in line 4.

/* SPECIFICATION OF MINILINK */

SEC HEADER

```

1  MODULE: MINILINK;
2  SOURCE: A.BANK,B.BANK,A.INPUT,B.INPUT,TRADAT;
3  TARGET: SOLUTION;
4  REFER: A.BANK,B.BANK,A.INPUT,B.INPUT,A.SOLUTION,
          A.A_STRUEONS,B.B_STRUC_EONS;

```

SEC DATA_DESCRIPTION

```

DISK IS MEDIA(UNIT=3330);

5  TRADAT IS FILE(DISK,ORG=SEQ,FORMAT=FIXED);
6  SHARE IS RECORD(TRADAT);
7  A70 IS FIELD(SHARE,(NO_CNTRY,NO_CNTRY,4));

/* SUBSCRIPT DESCRIPTIONS */

8  T IS SUBSCRIPT(PERIOD,A.LINK_YR,END_YR,1); /*SOLUTION
                                           TIME*/

9  DT IS SUBSCRIPT(DATA,1,20,1) /*DATA TIME*/
10 I IS SUBSCRIPT(A70,1,NO_CNTRY,1,1); /*COUNTRY*/
11 J IS SUBSCRIPT(A70,1,NO_CNTRY,1,2); /*COUNTRY*/
12 K IS SUBSCRIPT(A70,1,4,1,3); /*SITS CATEGORY*/

```

Figure 7.5

Specification of MINILINK


```

/* EACH COUNTRY ENDOGENOUS INTERIM VARIABLES */
13 A.VM,B.VM,A.PX,B.PX ARE INTERIM
      (DEC(14,5),(LINK_YR:END_YR,4));
14 A.X,B.X,A.PM,B.PM ARE INTERIM
      (DEC(14,4),(LAG_YR:END_YR,4));

/* MINILINK ENDOGENOUS INTERIM VARIABLES */
15 XL IS INTERIM(DEC(14,5),(NO_CNTRY,LAG_YR:END_YR,4));
16 VML,PHL,VXL,PXL ARE INTERIM
      (DEC(14,5),(NO_CNTRY,A.LINK_YR:END_YR,4));
17 VML09 IS INTERIM(DEC(14,5),(NO_CNTRY,A.LINK_YR:END_YR));

```

Figure 7.5 (continued)

SEC ASSERTIONS

/* DATA REPETITION ASSERTIONS */

18 END_YR = A.LINK_YR + A.LINK_NP;

19 LAG_YR = A.LINK_YR - 1;

20 NO_CNTRY = 2;

/* ASSERTIONS LINKING EACH COUNTRY WITH MINILINK */

21 VML(1,T,K) = A.VM(T,K);

22 VML(2,T,K) = B.VM(T,K);

23 PXL(1,T,K) = A.PX(T,K);

24 PXL(2,T,K) = B.PX(T,K);

/* ASSERTIONS LINKING MINILINK TO EACH COUNTRY */

25 A.X(T,K) = XL(1,T,K);

26 B.X(T,K) = XL(2,T,K);

27 A.PM(T,K) = PML(1,T,K);

28 B.PM(T,K) = PML(2,T,K);

Figure 7.5 (continued)

```

/* ASSERTIONS TO INCLUDE INITIAL LAGS */
29 A.X(LAG_YR,K) = A.DATA(5,K,LAG_YR);
30 B.X(LAG_YR,K) = B.DATA(5,K,LAG_YR);
31 A.PM(LAG_YR,K) = A.DATA(1,K,LAG_YR);
32 B.PM(LAG_YR,K) = B.DATA(1,K,LAG_YR);

/* ASSERTIONS FOR ENDOGENOUS LINKAGE VARIABLES */
33 VXL(I,T,K) = SUM(A70(I,J,K)*VML(I,T,K),J);
34 PML(J,T,K) = SUM(PXL(I,T,K)*A70(I,J,K);
35 XL(I,T,K) = VXL(I,T,K)/PXL(I,T,K);

/* ASSERTIONS FOR SOLUTION VARIABLES */
36 PERIOD(T) = T;
37 NAME(T) = 'NAME';
38 PRICE(T) = 'PRICE IMPORT';
39 VALUE(T) = 'VALUE IMPORT';
40 CTRY_NAME(T,I) = I;
41 VML00(I,T) = SUM(VML(I,T,K),K);
42 P(I,I) = SUM(PML(I,T,K)*VML(I,T,K)/VML00(I,T),K);
43 V(T,I) = SUM(VXL(I,T,K),K);
44 TZ(T) = SUM(V(I,T),I);

```

Figure 7.5 (continued)

7.4.3 Summary of Specification of Integrated Models

As shown each individual country model specification as well as the specification of MINILINK are entered into the MODEL specification library. Each one of these modules when submitted to the MODEL system will result in generation of a computer program optimized for the specific purpose of simulating the respective model. In the example above, programs for the simulation of Country "A", "B" and MINILINK will be obtained. Each of these specifications required approximately 50 statements. The resulting PL/I programs would consist of an order of magnitude larger number of statements.

Note that the specifications could be composed incrementally with the system addressing messages to the user to indicate additions needed and the user responding by composing the statements that are required. Also many of the statements shown could be generated automatically based on analysis of the previously entered statements. The documentation of the specifications to be produced by MODEL will include many additional statements for each one of these modules, that can be used to further verify the intention of the user.

Upon submission of the TRADE module for execution by the MODEL processor, the complete specification will be analyzed and simultaneous assertions reported to the user. It will be then necessary to specify the initial values and

convergence criterion. Appendix C shows the complete listing of the MINILINK example, including the source listing and MODEL generated statements, those assertions found to be simultaneous and the formatted report. Also included are the cross-reference and precedence reports.

The linkage algorithm described by the assertions in Figure 7.5 is similar to the implementation of the "constant value shares" version as described in [GAN 76b]. For other linkage algorithms (ie LES), it is necessary to implement a function RAS, to perform a standard RAS adjustment. Such function can be implemented with the following parameters:

RAS(M,R,C)

M = Matrix with initial conditions in each element $M(i,i)$. After processing M will contain the adjusted elements $M(i,i)$.

R = "row totals" vector

C = "column totals" vector

CHAPTER 8

Using MODEL for Regression and Statistical Analysis

8.1 General Approach

Within the methods of statistical inference, estimation of parameters and subsequent tests of significance of these estimates play a crucial role in model building activity. The advantages of using a non-procedural language in this stage are similar to those previously stated for the generation of simulation and data transformation programs in general. While a variety of advanced statistical techniques have been developed and experimented with lately, it is still difficult for an econometrist to obtain easy access to their computer implementation. It is even more difficult to combine existing procedures into a consistent estimation system.

In general existing estimation software falls into two categories; the first one consisting of high level interactive or command oriented languages which are user oriented but whose procedures are given as "black boxes" which, except for the straightforward methods, may cause problems of interpretation to the unaware user. This can generally be solved with proper documentation. However a more serious disadvantage is their lack of flexibility to incorporate new techniques or to combine existing ones. Typical examples of languages in this category are PLANETS

and TSP. The second group include those programs developed to carry out specific estimation procedures and are normally written in a high level procedural language such as FORTRAN, APL or PL/I. These packages are normally developed "in-house" by different modeling centers in order to solve a particular application need, consequently there is lack of standarization and therefore it is difficult to combine or integrate them without a substantial labor intensive investment. A programming language is not the natural media for an econometrist to represent his algorithms, hence it is also difficult for him to unveil the semantics of a computer procedure or to communicate about it with a programmer. Most estimation techniques can be constructed around a reduced set of basic building blocks; however, because of the individualistic approach to programming via procedural languages, substantial duplication exists in different centers using stand alone packages for their estimation and statistical analysis requirements.

A notable exception to the above mentioned characteristics of most estimation kits or languages to be considered is the case of the NBER TROLL system [TRO 73], where a coherent and consistent set of estimation procedures has been implemented using a generalized approach [EIS 73]. The integrated system is built around basic operations, which are functionally organized as separate entities, which can be combined together in conjunction with an ordering procedure to provide meaningful statistics. TROLL functions

are built around a least squares procedure, sets of data transformations and statistical analyses. They include:

- 1.- An OLS procedure for either linear or nonlinear equations.
- 2.- Distributed lag operators.
- 3.- Instrumental variable estimation.
- 4.- Single equation Generalized Least Squares (GLS) error correction.
- 5.- A standard set of statistics based on the observed residuals as well as the coefficient covariance matrix.

Proper selection and combination of these procedures suffices to produce standard single equation techniques such as TSLS or autoregressive adjustments.

The basic ordering used when different procedures are combined is as follows:

- (1) Generalized Least Squares transformations.
- (2) Instrumental variable substitution.
- (3) Polynomial distributed lag operators.
- (4) OLS as applied to linear or nonlinear equations.

The approach to estimation followed by MODEL is twofold: first it extends the language by introducing the necessary primitive functions and/or operators which will allow the user to generate the desired estimation program from the mathematical description of the procedure or algorithm.

Secondly, some estimation procedures can be directly implemented to operate into a single or simultaneous set of assertions by proper identification of the methods using the SOLUTION header parameter of an assertion.

By extending the language through functions and/or operators, MODEL can be made isomorphic with the algebraic language with which a model builder is accustomed to represent his computation requirements. It is therefore possible to generate any desired estimation program from the user non-procedural specification. The second approach is more user oriented, in the sense that it permits him to keep visible the structural representation of the functions whose parameters he wishes to estimate, but the process itself is transparent. In this latter case the processor is responsible for checking the consistency and completeness of parameters and solution procedures required.

This mixed approach is expected to give greater flexibility and generality as well as savings in the software development process. The next sections describe the necessary extensions and design methodology.

3.2 Some Useful Functions

Model building, statistical and regression analysis as well as Input-Output computations make extensive use of matrix algebra in describing concisely its calculations. Therefore it is convenient to extend the scalar

characteristics of MODEL to cope with general matrix and vector operations. Ideally one will like to include vector and matrix operators such as those found in the APL programming language [IVE 62], but in the non-procedural context of the MODEL processor. The following basic list of MODEL matrix manipulation functions can be used for that purpose:

Function: $R = \text{MATMLT}[D] (A, B, K, L, M) *$

Purpose: Matrix A is postmultiplied by matrix B. The standard matrix product $R = A \cdot B$ is performed, i.e., element $R(i, k)$ is the scalar product of the i th row of A with the k th column of B.

Arguments:

$A(K, L) - \text{BINARY FLOAT}[(53)]$; given K by L matrix A
(left-hand factor).

$B(L, M) - \text{BINARY FLOAT}[(53)]$; given L by M matrix B
(right-hand factor).

K - BINARY FIXED; row dimension of A and R.

L - BINARY FIXED; column dimension of A and row dimension of B.

M - BINARY FIXED; column dimension of B and R.

Returns:

* Optional expressions in square brackets "[]" indicate the double precision alternative function. The code generation phase should produce necessary attribute declarations for user supplied parameters when they are not in agreement with the arguments of the function and a transformation is possible. This is done via the ENTRY statement.

R(K,M) - BINARY FLOAT[(53)]; resultant K by M product matrix.

Function: R = MINV[D] (A,N,ZERO)

Purpose: Inverts a general square matrix A.

Arguments:

A(N,N) - BINARY FLOAT[(53)]; given N by N general matrix A.

N - BINARY FIXED; order of matrix A.

ZERO - BINARY FLOAT[(53)]; given constant with which the determinant is compared for singularity test. If the given value is zero, the function assigns the value 10E-5 if single precision or 10E-15 if double precision is used.

Returns:

R(N,N) - BINARY FLOAT[(53)]; resultant square inverse matrix of A.

Function: R = TRANSP[D] (A,M,N)

Purpose: Transposes matrix A.

Arguments:

A(M,N) - BINARY FLOAT[(53)]; given M by N matrix to be transposed.

M - BINARY FIXED; row dimension of A and column dimension of R.

N - BINARY FIXED; column dimension of A and row dimension of R.

Returns:

R(N,M) - BINARY FLOAT[(53)]; the transpose of A.

Function: $R = \text{EIGVAL}[D] (A, N)$

Purpose: Computes eigenvalues of symmetric matrix A.

Arguments:

$A(N, N)$ - BINARY FLOAT[(53)]; given square symmetric matrix.

N - BINARY FIXED; order of matrix A.

Returns:

$R(N)$ - BINARY FLOAT[(53)]; eigenvalues of A are returned in vector R.

Function: $R = \text{EIGVEC}[D] (A, N)$

Purpose: Computes eigenvectors of symmetric matrix A.

Arguments:

$A(N, N)$ - BINARY FLOAT[(53)]; given square symmetric matrix.

N - BINARY FIXED; order of matrix A.

Returns:

$R(N, N)$ - BINARY FLOAT[(53)]; eigenvectors of matrix A are returned in matrix R.

Function: $R = \text{MOMENT}[D] (A, M, N)$

Purpose: Computes moments of matrix A. That is calculates $R = A'A$. This function takes advantage of symmetry found in the computation of moments.

Arguments:

$A(M, N)$ - BINARY FLOAT[(53)]; the given M by N matrix, eq. the augmented matrix $[y, X]$ of dependent and independent observations for computing OLS estimation.

M - BINARY FIXED; row dimension of A, eg. the number of

observations for time series.

N - BINARY FIXED; column dimension of A, eg. the total number of variables for moment calculations.

Returns:

R(N,N) - BINARY FLOAT[(53)]; the square symmetric matrix of moments $R = A'A$.

Function: R = MMVERT[D]: (A,N,ZERO)

Purpose: Obtains inverse of symmetric matrix of moments (A), and computes OLS coefficients $B = INV(X'X)*X'Y$.

Arguments:

A(N,N) - BINARY FLOAT[(53)]; Input matrix of moments.

ZERO - BINARY FLOAT[(53)]; tolerance constant with which the determinant is compared for singularity test.

N - BINARY FIXED; dimension of square matrix A.

Returns:

R(N,N) - BINARY FLOAT[(53)]; Inverse of A with OLS coefficient vector "B" in R(1:N-1,N).

Function: R = MATVEC[D] (A,B,M,N)

Purpose: Matrix A is postmultiplied by vector B to yield vector R. Matrix A must have as many columns as vector B has elements.

Arguments:

A(M,N) - BINARY FLOAT[(53)]; given M by N matrix A (left-hand factor).

B(N) - BINARY FLOAT[(53)]; given vector B of M elements (right-hand factor).

M - BINARY FIXED; row dimension of A and number of items in R.

N - BINARY FIXED; column dimension of A and number of elements of B.

Returns:

R(M) - BINARY FLOAT[(53)]; resultant product vector of M elements.

Notice that all the previous MODEL functions involve a more general concept than the regular PL/1 function, in the sense that the latter restricts its outcome to a single value, whereas MODEL functions can return a matrix or a more general array structure. To accomplish this the code generation phase must recognize this special MODEL functions and generate a corresponding procedure CALL in place. For example for the function MOMENTD the following procedure call is generated:

CALL MOMENTD (R,Z,NO,NV);

and the following procedure code is included;

```

MOMENTO: PROCEDURE (R,A,M,N);
  DECLARE (A(*,*),R(*,*))
    BINARY FLOAT(53),
    (M,N,I,J,K) BINARY FIXED,
    S BINARY FLOAT(53);
  DO I = 1 TO N;
    DO J = 1 TO N;
      S = 0.;
      DO K = 1 TO M;
        S = S + A(K,I)*A(K,J);
      END;
      R(I,J),R(J,I) = S;
    END;
  END;
  RETURN;
END MOMENTO;

```

Subsequently μ is substituted in place of the MOMENTO function reference in the code. Also an ENTRY statement should be generated to identify the external procedure and must include an attribute parameter list if there is no matching between parameters and arguments.

Other functions used in model building were added to the MODEL library. Some typical examples of functions used to generate random variates are:

$$X = \text{UNIFORM}(A,B)$$

to generate uniformly distributed variates in the range between A and B;

$$X = \text{EXPON}(EX)$$

to generate exponentially distributed variates with mean EX;

$$X = \text{GAMMA}(K,A)$$

for gamma distributed variates with parameters K and A;

$$X = \text{NORMAL}(EX,STD\!X)$$

for normally distributed variates with mean EX and standard deviation STD $\!X$. These as well as other general matrix and vector manipulation functions implemented in the MODEL library are listed in Appendix D.

8.3 Examples Using Functions

8.3.1 An Input-output Matrix

This first example illustrate how to describe a module to form an input-output Leontief matrix $(I - A)^{-1}$. 'A' is a source matrix of technical coefficients size N by N

```
/* FORMING THE IDENTITY MATRIX "I" */
  I IS INTERIM (BIN FIXED, (N,N));
  K,L ARE SUBSCRIPT (1,N);
  IF K=L THEN I(K,L) = 1
  ELSE I(K,L) = 0;
/* SUBTRACT MATRICES */
  S = I - A;
/* OBTAIN THE INVERSE */
  X = MINV(S,N,0);
```

8.3.2 Ordinary Least Squares Regression Module

OLS calculations can be specified in a number of ways using some of the functions previously described. The expression for the vector \hat{B} of estimated coefficients is:

$$\hat{B} = (X'X)^{-1} X'Y$$

where $Y[T]$ is the vector of observations for the dependent variable and $X[T,N]$ is the matrix of observations for the N explanatory variables. The vector of coefficients is defined as $B[N]$.

Let

```

XT IS INTERIM (N,T);
MX IS INTERIM (N,N);
MY IS INTERIM (N);
MI IS INTERIM (N,N);

```

Then

```

/* THE TRANSPOSE OF X'X IS PLACED IN 'XT' */
XT = TRANSP(X,T,N);

/* MOMENTS X'X AND X'Y ARE COMPUTED AND STORED IN 'MX'
AND 'MY' RESPECTIVELY */
MX = MATMLT(XT,X,N,T,N);
MY = MATVEC(XT,Y,N,T);

/* INVERSE(X'X) IS KEPT IN 'MI' */
MI = MINV(MX,N,0);

/* COMPUTE REGRESSION COEFFICIENTS */
B = MATVEC(MI,MY,N,T);

/* FORM XB, THE PREDICTED VALUES IN YHAT */
YHAT IS INTERIM (T);
YHAT = MATVEC(X,B,T,N);

/* OBTAIN RESIDUALS Y - XB in R */
R IS INTERIM (T);
R = Y - YHAT;

```

Finally some statistics can be computed as follows: The variance of the estimated residuals, VAR, is given by

```

VAR = (1/(T-N)) * SUM(R(K)**2,K);
K IS SUBSCRIPT (1,T);

```

and R^{*2} , adjusted for degrees of freedom, is given by

$$RSQ = 1 - VAR/SIGMSQ;$$

and

$$SIGMSQ = (1/(T-1)) * (SUM(Y(K)**2,K) - SUM(Y(K),K)**2);$$

The OLS specification may appear extremely complex for a prospective user. However the concept that must be insisted on is that it is not a program to be used ordinarily for OLS computations, it is a description for the design of a program which will be generated to carry out such computations. The econometrist is expected to provide definitions with which he is familiar. He must also provide the necessary description for input and output files. If in providing this information requirements, careful considerations are given to the user interface, then the user who at a lower level is going to execute the generated program will have only to provide the required data for evaluation (for instance from a terminal and in free format).

8.4 Direct Implementation of Estimation Methods

This section explores the feasibility and implications of a direct implementation of estimation techniques, for either single or simultaneous equations, using the existing storage structure for assertions in the AI of the MODEL processor. The discussion is centered on the necessary methodology for the embedding of a general and complete set of estimation techniques into the non-procedural nature of

MODEL. For detailed background on methods and algorithms used in estimation, the reader is referred to standard econometric textbooks such as Johnston [JOH 72] or Klein [KLE 74].

The general approach used by TROLL in single equation estimation is well suited for implementation into MODEL. The main characteristics of this approach can be summarized as follows:

1.- For OLS, an equation of the form

$$f(X, B) + e = 0$$

where $X=(x_1, \dots, x_m)$ is a set of data vectors, 'B' is a set of parameters, 'f' is a function of these two data sets and 'e' is an implicit additive error vector, is linearized using a Taylor Series expansion so that it fits the form $(Y - XB) = e$. This results in a quadratic objective function to be minimized

$$\hat{e}'\hat{e} = (Y - X\hat{B})'(Y - X\hat{B})$$

where ' \hat{e} ' represents the observed residuals and ' \hat{B} ' the estimated set of parameters. An explicit solution for ' \hat{B} ' results in the familiar first order conditions

$$\hat{B} = (X'X)^{-1} X'Y$$

The advantage of this approach is that in the case of a linear equation, it reduces directly to the calculation of ordinary least squares. In the case of nonlinear equations, the Taylor series expansion must be made around some initial values of 'B', call it B^0 . This involves an iterative

process; at each step a regression is performed on the linearized equation, and a test is made to see if the computed \hat{B} are different from B^0 . If the test criterion is not met, B^0 is set equal to \hat{B} and the steps are repeated.

From the user point of view, it has the added advantage that it is not required to enter a linear equation in the canonical sum of products form.

2.- OLS can be combined with other correction techniques. For example instrumental variables transformation or generalized least squares can be combined with a linear regression in the standard manner. For nonlinear regression, the techniques are applied at each iteration step on the expanded linear equation form.

3.- Distributed lag equations of the form

$$\sum_{\tau=0}^n p(a_{\tau}, x_{t-\tau}) + f(x_t, \beta) + e = 0$$

can be estimated by assuming that the estimates of ' a_{τ} ' are produced by an l th degree polynomial

$$a_{\tau} = \sum_{j=0}^L w_j \tau^j$$

This method of constraining the estimates of ' a_{τ} ' in the lag operator to reduce the problem of collinearity is known as the Almon process.

4.- For a Generalized Least Squares correction, given the general equation form

$$f(x, \beta) + e = 0$$

in which the covariance matrix of ' e ' is known

$$E(ee') = \sigma^2 V$$

a transformation matrix 'A' is constructed by factoring ' V^{-1} ', i.e., 'A' is formed such that

$$A'A = V^{-1}$$

The equation is linearized in the usual way, and then it is premultiplied by the transformation matrix 'A'. The ordinary least squares procedure is subsequently applied to the transformed equation.

The generalized least squares procedure can be used in conjunction with a symbolic transformation 'A' to correct for first or second order autoregressive processes of the form

$$e(t) = \rho_1 e(t-1) + e^*(t) \text{ for first order, or}$$

$$e(t) = \rho_1 e(t-1) + \rho_2 e(t-2) + e^*(t) \text{ for second order}$$

In these cases the error covariance matrix and its inverse will be a function of the parameters ρ_i , and again an iterative schema similar to the familiar Hildreth-Lu [HIL, LU 60] process will result. In cases of nonlinear equations two iterative processes are involved.

5.- As mentioned previously in section 8.1, a logical ordering procedure is applied to resolve or best insure consistency and efficiency in the estimates when multiple methods and or transformations are applied.

After this brief introduction to a general approach to estimation, as used in the TROLL system, assume a user of MODEL wants to estimate an equation such as

$$EQ1: Y = (1 - A) * X + B + ERROR;$$

Since EQ1 is a linear assertion, after the linearization procedure it will be transformed into the expression

$$EQ1: Y - X = - A * X + B + ERROR;$$

to which the OLS regression method can be applied. For instance the user can identify the method and parameters through the SOLUTION statement header for an assertion

example

EQ1:

SOL: OLS, A^0, B^0

$$Y = (1 - A) * X + B + ERROR;$$

where (A^0, B^0) is a given initial guess for the coefficient vector, to be used in nonlinear cases, and set to zero for linear equations.

First it must be noticed that the usual SOURCE, TARGET classification of variables in MODEL is not sufficient for estimation purposes. In the case of regression analysis the TARGET variables actually are the parameters (A, B) , but the user normally wants to maintain the structure of his equation depicting the dependent and explanatory variables. Notice also that the dependent variable can imply a temporary transformation for purposes of estimation only (e.g. $Y - X$ in EQ1). The ERROR or residual term can also be considered a TARGET type variable, since it will be computed by the regression method selected to produce appropriate statistics. If MODEL is extended to include the data type

PARAMETER, it is possible to determine automatically the necessary transformations and generate the required estimation program

example:

```
A,B ARE PARAMETERS (FIXED(10,4));
```

```
EQ1:
```

```
SOL:OLS
```

```
TARGET:ERROR,A,B
```

```
SOURCE: Y,X
```

```
Y = (1 - A)*X + B + ERROR;
```

The processor will not normalize to ERROR since it will recognize the solution method as requiring regression analysis and will proceed to the linearization process yielding

```
Y - X = -A*X + B + ERROR;
```

as the final form. Also an INTERIM variable $I = Y - X$ will be formed to hold the transformed dependent variable for the regression program.

It is also possible to identify the special nature of coefficients (A,B) as part of the solution parameters (i.e., SOL: OLS,A,B), or identify EXPLANATORY and or DEPENDENT variables in the header of the assertion.

Since the cycles processor in MODEL can identify the recursive structure of a system composed of simultaneous

assertions, it is possible to extend the single equation estimation methodology to systems of simultaneous equations by including Full Information Maximum Likelihood (FIML), 3SLS or Iterative Least Squares (ILS) methods. The basic mechanisms to generate iterative solution programs, at two levels, is already implemented in MODEL, however the Completeness Analysis should be extended to take account of possible new data types in the assertions and more importantly, the Sequencing phase should analyse the ordering of methods of estimation and its transformations to produce meaningful statistics.

8.5 Further Extensions of Single Equation Methods

The basic primitives used to generate single equation regression programs (OLS, TSLS or LIML) can also be used to produce modules for estimation of principal components and Ridge regressions. Similarly it is possible to modify the Almon procedure for distributed lag models and include Shiller's method derived from smoothness priors [SHI 73]. These are new methods which are being empirically tested as possible solutions to the problem of multicollinearity, therefore their inclusion in a modeling system should be considered. Following is a brief description of their properties and characteristics for implementation.

8.5.1 Principal Components

Principal components of a set of predetermined variables $X = x_{1t}, x_{2t}, \dots, x_{mt}$ are defined as mutually orthogonal, linear combinations of the elements of X

$$pc_{it} = \sum_{j=1}^m \delta_{ij} x_{jt} \quad i=1,2,\dots,r; \quad t=1,2,\dots,T$$

such that pc_{it} has maximum variance for $\delta_{i1}^2 + \delta_{i2}^2 + \dots + \delta_{im}^2 = 1$ (the normalization condition), and the pc_{it} are mutually uncorrelated (independent).

The principal components are estimated as the set of characteristic vectors corresponding to the characteristic roots of

$$|X'X/T - \lambda I| = 0$$

Instead of the moment matrix $X'X$, it is possible to use the correlation matrix "R" as standardized form with

$$r_{ij} = \frac{\sum_{t=1}^T (x_{it} - \bar{x}_i)(x_{jt} - \bar{x}_j)}{\left[\sum_{t=1}^T (x_{it} - \bar{x}_i)^2 \sum_{t=1}^T (x_{jt} - \bar{x}_j)^2 \right]^{1/2}}$$

and $\bar{x}_i = (1/T) \sum_{t=1}^T x_{it}$

The first (largest) "r" roots whose characteristic vectors (components) account for a preassigned percentage of the variance of "X" are selected as instruments. "Y" is then regressed on the reduced set of principal components pc_{1t}, \dots, pc_{rt} , where $T > r$. The pc_{it} can be used in the first stage regression of a TSLS or LIML, where instead of computing the inverse $(X'X)^{-1}$, the calculation involves the inverse $(P'P)^{-1}$.

From the computation point of view, it is not necessary

to evaluate implicitly the principal components as such, since moment calculations with linear combinations of variables can be expressed as linear combinations of the original variables as follows:

In terms of the data matrix of predetermined variables, the P matrix is

$$P = X \Gamma' \quad (8-1)$$

therefore the moments are

$$P'P = \Gamma'X'X\Gamma \quad (8-2)$$

The EIGEN[D] procedure in Appendix D uses the Power method to obtain all the eigenvalues and eigenvectors of an nxn real matrix A. The same procedure can easily be modified to compute only the first (largest) characteristic roots of the equation

$$|R - \lambda I| = 0$$

and terminate once enough components have been computed to account for the preassigned percentage of the general variance. The ratio of $\hat{\lambda}_i$ to the sum of the diagonal elements of R (TRACE(R)) that are normalized to unity in each case, gives the percentage of variance explained by the ith component. The cumulative total of the percent variance accounted for by the first "r" pc's is given by $\sum_{i=1}^r \hat{\lambda}_i / m$.

After evaluation of Γ , the module should form the right hand side of expression (8-2), using the MOMENT[D] function in calculation of the original moment matrix plus regular

matrix multiplication and transposition (MATMLTD, TRANSP). The resulting matrix will be diagonal, since the principal components are mutually orthogonal, therefore the inverse of it could be obtained by a simpler inverse function.

$$(P'P)^{-1} = \begin{vmatrix} 1/\sum p_{1t}^2 & 0 & \dots & 0 \\ 0 & 1/\sum p_{2t}^2 & \dots & 0 \\ . & . & . & . \\ 0 & \dots & \dots & 1/\sum p_{rt}^2 \end{vmatrix}$$

8.5.2 Shiller's Method and Ridge Estimators

In a distributed lag model

$$y_t = \sum_{i=0}^p \beta_i x_{t-i} + \mu_t \quad (8-3)$$

with $\mu_t \sim IN(0, \sigma_\mu^2)$, $t=1, 2, \dots, n$

the Almon method assumed that the β_i lie in an l th degree polynomial, then

$$\beta_i = \sum_{j=0}^l \alpha_j \tau^j \quad (8-4)$$

For illustration purposes assume a quadratic polynomial, then

$$\beta_i = \alpha_0 + \alpha_1 \tau + \alpha_2 \tau^2 \quad (8-5)$$

and equation (8-3) can be written as

$$\begin{aligned} y_t &= \sum_{i=0}^p (\alpha_0 + \alpha_1 \tau + \alpha_2 \tau^2) x_{t-i} + \mu_t \\ &= \alpha_0 z_{0t} + \alpha_1 z_{1t} + \alpha_2 z_{2t} + \mu_t \end{aligned} \quad (8-6)$$

and

$$z_{jt} = \sum_{i=0}^p \tau^j x_{t-i} \quad (8-7)$$

Equation (8-4) can be written as

$$\beta = H \alpha \quad (8-8)$$

and for the quadratic case

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \\ \vdots & \vdots & \vdots \\ 1 & P & P^2 \end{bmatrix}$$

Define $M = I - H(H'H)^{-1}H'$, then (8-8) imply the restriction

$$M\beta = 0 \quad (8-9)$$

Thus the Almon estimator minimize $(Y - X\beta)'(Y - X\beta)$ subject to the restriction in (8-9). This gives

$$\hat{\beta}_A = \hat{\beta} - (X'X)^{-1}M'[M(X'X)^{-1}M']^{-1}M\hat{\beta} \quad (8-10)$$

where $\hat{\beta}_A$ is the Almon estimator of β and $\hat{\beta}$ represents the OLS estimator of β .

Shiller notes that (8-5) implies

$$\Delta^3 \beta_i = 0; \quad \Delta \beta_i = \beta_i - \beta_{i-1} \quad (8-11)$$

and further makes (8-11) stochastic by adding an error term $w_i \sim IN(0, \sigma_w^2)$, that is $\Delta^3 \beta_i = w_i$

or

$$R\beta = w \quad (8-12)$$

where

$$R = \begin{bmatrix} 1 & -3 & +3 & -1 & 0 & 0 & \dots\dots\dots 0 \\ 0 & 1 & -3 & +3 & -1 & 0 & & & & 0 \\ . & . & . & . & . & . & & & & . \\ . & . & . & . & . & . & & & & . \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & -3 & +3 & -1 \end{bmatrix}$$

R is a $(p-1) \times (p+1)$ matrix. The estimator of β is given by

$$\hat{\beta}_s = (X'X + kR'R)^{-1} X'Y \quad (8-13)$$

where $k = \sigma_u^2 / \sigma_w^2$ is assumed known.

Hoerl and Kennard [HOE 70] suggested a set of biased least squares estimators which they call "ridge estimators". Instead of the OLS estimator

$$\hat{\beta} = (X'X)^{-1} X'Y \quad (8-14)$$

they propose the modified estimator

$$\hat{\beta}_R = (X'X + kI)^{-1} X'Y \quad (8-15)$$

Again K is assumed known, or must be computed from some iterative schema. A survey on the properties of these estimators and related techniques is given in [VIN 78]. A more general form can be considered

$$\hat{\beta}_R = (X'X + kQ)^{-1} X'Y \quad (8-16)$$

where "Q" is a positive semidefinite matrix, then it can be seen that Shiller's estimators as defined in (8-13) are also ridge estimators.

The main problem in calculating these estimators is

determining a suitable value for "k". Lindley and Smith [LIN 72] suggest an iterative method which can be applied to both ridge and Shiller estimators. The process start with $\hat{\beta}_i$, the OLS estimate of β_i , and then proceeds by taking the variance of the estimates as an estimate of σ_w^2 :

$$\hat{\sigma}_w^2 = (1/n) \sum_{i=0}^p (\beta_i - \bar{\beta})^2$$

After the new estimates are computed, the estimate of $\hat{\sigma}_w^2$ is revised based in the new estimates. The estimate of σ_μ^2 are taken from the least squares residuals. The procedure should stop when some desired preset convergence criteria is obtained.

MODEL provides the basic matrix and vector operation functions to non-procedurally describe these techniques, as well as the iterative mechanism for their solution.

CHAPTER 9

Conclusions

The MODEL system described in the preceding chapters permits the automatic generation of model building application programs from a non-procedural specification in the MODEL language. The techniques and facilities introduced by this research illustrated the feasibility of such an approach. Continuation of present trends with respect to usage, size and complexity of different models used in the physical, social and natural sciences indicate the proposed new methodology as a viable and cost effective alternative for software development as well as for providing the model builder with a direct communication interface for his structural ideas. Those aspects of MODEL which permit the sharing and integration of different models and their data structures were also demonstrated. These facilities will allow the incremental generation of knowledge databases encompassing the contributors expertise.

It is premature yet to draw comparisons with regard to the quality and efficiency of the code generated by the MODEL processor, since the code generation and global optimization phases are still in developmental and research stages. However it is possible to infer from comparisons performed in previous versions of MODEL, as applied to business data processing examples [RIN 76], that the efficiency of a program generated by the processor will be

good as compared with a manually written program. The fact that the user can generate ad-hoc programs for particular problems also contribute to the efficacy of the generated programs, which otherwise have to bear unnecessary the overhead code incurred by general purpose modeling packages.

Although at first glance the specification for Klein's Model I given in Appendix A may appear complex in relation to the statements required for solving such model in other systems, a careful evaluation suggest the opposite. Since existing modeling software do not posses generalized data description capabilities, this is always implicit in the system and imposed to the user. With MODEL the user can create or access a library of appropriate specifications for different data structures, and needs only to refer to them for inclusion in his specification. In such case, the model builder should concern himself only with the description of the model structural equations. The same concepts can be extended to ad-hoc description of reports, once they are in the user's data base, different modules can refer to them for inclusion in the respective specifications. In addition, notice that over thirty percent of the statements in the complete specification list were actually generated by the MODEL system. These considerations, in addition to the tolerance provided by the processor, strongly favor it in relation to other systems, particularly as the complexity and information requirements of a given model grows.

In summary, the MODEL language and its processor promise to be a valuable tool for developing the software needed at different stages by large models. While several refinements of the existing system are possible, it is likely that the greater benefits will come from actual testing of the system in the hands of model builders. A dynamic environment requiring communication, integration and solution of models of different sizes and complexities will probably contribute to the basic system presented in this dissertation, and an enhanced knowledge database and system will emerge from such interaction.

Several extensions to the present system are possible, and some of them were mentioned in previous chapters. The most obvious are concerned with generation of new solution procedures and algorithms, as exemplified in Chapters 6 and 8. Although the language is open-ended for procedural applications through the inclusion of appropriate functions, in many instances it is possible to delegate more deductive power to the processor, such as to automatically determine those procedural needs and generate appropriate algorithms after analysis of the given specification (and possible interaction with the user) shows that a procedural solution method is needed to avoid inconsistencies. Examples of such analysis and interaction are the cycles processor for simultaneous equations, and the proposed general approach to estimation as described in Chapter 8. Several other applications could benefit from such approach. For

instance, if the syntax of MODEL is extended to include simple assertions with inequalities representing constraints, then the cycles algorithm could be used to identify linear or non-linear programming problems in which an objective function should be maximized (minimized) subject to some related inequalities. For these problems it is important to identify possible rearrangement into "block angular" form, as shown in Figure 9.1, to subsequently use any of the well known decomposition algorithms [WET 71].

$$\begin{array}{cccc}
 A(0,1) & A(0,2) & \dots\dots & A(0,n) \\
 A(1,1) & 0 & \dots\dots & 0 \\
 0 & A(2,2) & \dots\dots & . \\
 . & . & \dots\dots & . \\
 . & . & \dots\dots & . \\
 . & . & \dots\dots & . \\
 0 & 0 & \dots\dots & A(n,n)
 \end{array}$$

Figure 9.1
 Canonical Block Angular Form
 ($A(i,j)$ are arbitrary rectangular matrices)

Different maximization algorithms such as those used in optimal control applications of large econometric models [AND 76] can also be implemented and the domain of applicability of MODEL will therefore be expanded. Notice that it is possible to use the present system for simple optimal control studies by augmenting the implicit definition of a model with additional relations derived from the maximization conditions of a welfare or objective function. However a more sophisticated system should be able to perform symbolic differentiation on the welfare function with respect to the policy variables in all time periods, and automatically generate additional assertions to endogenize selected policy variables. In a typical dynamic model which uses long distributed lags, the total number of equations and variables can increase dramatically when the system is converted to first order, therefore the advantages of mechanizing the procedure.

Chapter 5 discussed a number of possible extensions to the basic cycles algorithm, in order to "tear" or further decompose a set of strongly connected components into smaller and more manageable groups. The problems of partition, near-decomposability and aggregation have been shown to be of relevance in social sciences, and therefore it will be valuable to extend the system in those directions such as to help the user to better understand the interconnections and complexities of his system. Finally Chapter 7 presented a basic mechanism for implementation of

different algorithms for the automatic selection of "output sets" or rules of normalization. Applications in the physical sciences and engineering will probably benefit by introducing such facilities.

Some Experimental techniques to check the consistency of a model specification can also be added. For instance Steuert [STE 74] suggest a graph theoretical algorithm to generate all "submodels" of a simultaneous equation model, and then test them for convergence and use Fisher's correspondence principle [FIS 70] to locate specification problems. The cycles processor could be extended to include this and other algorithms as alternative solution procedures.

In addition to the refinements and extensions above mentioned, perhaps the most worthwhile direction for further research would be to endeavor in the design and implementation of application oriented interfaces, whose tasks would be that of guiding the user in composing his specification in a language which is more natural to the application practitioner than the general syntax required by MODEL. From such higher level interaction, the required MODEL language specification can then be generated. Figure 9.2 illustrate a proposed system for economic modeling. The existing MODEL system is indicated in the center of the diagram, and can be considered to represent the "expert" system in programming and logic required to generate

particular software. The top box contain the "knowledge" for a given application, such as to guide the user in composing his statements, with possible preanalysis and interpretation of intermediate tasks. An economist will describe his data to the interface as being time-series or cross sectional, will identify the components of his model in terms of parameters, exogenous or endogenous variables and functional relationships. Solution procedures will be identified for different types of simulations, residual checks or policy analysis, etc.. A CALCULATOR mode could be provided for the creation and transformation of new variables and other simple statistical and data manipulations requiring immediate interpretation. Finally a complete specification in the MODEL language will be produced for processing by the Automatic Program Generator System.

While much research lies ahead in the field of automatic program generation, it is hoped that this research has contributed towards that end, increasing the generality of MODEL and providing a valuable tool for sharing and integrating the knowledge imbeded in the modeling process in a way that it becomes accessible to both humans and machines.

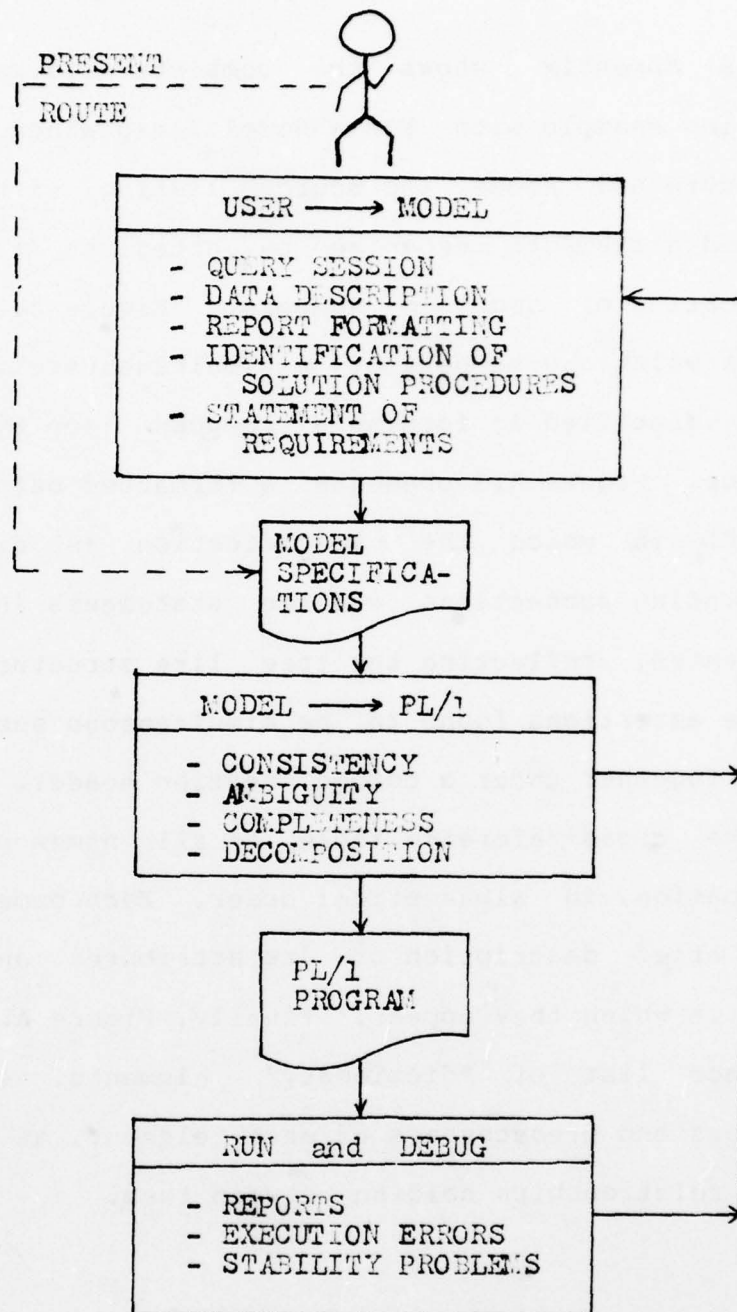


Figure 9.2
Proposed MODEL System Development

APPENDIX A

This Appendix shows the complete listing of the simulation example with Klein Model I explained in Chapter 3. Figure A.1 shows the source listing with the MODEL generated statements recognized by asterisks '*' printed in the location of sequence numbers. Figure A.2 list the elements which are members of a simultaneous equation group and are identified as forming a compound loop by the cycles processor. Figure A.3 presents a formatted output produced by MODEL in which the specification is divided into corresponding subsections and the statements in each file are indented, reflecting the tree like structure in them. Also the assertions found to be simultaneous assertions are grouped together under a common section header. Figure A.4 shows the cross-reference table of all names used in the specification, in alphabetical order. Each name is listed with a brief description of its attributes and the line numbers in which they appear. Finally, Figure A.5 gives the precedence list of "dictionary" elements, showing the successors and predecessors of each element, as well as the type of relationships holding between them.

SOURCE LISTING AND GENERATED STATEMENTS

STATEMENT NUMBER		
1	/******	00000010
1	/*	00000020
1	/* KLEIN MODEL 1 SPECIFICATION	00000030
1	/*	00000040
1	/******	00000050
1	/******	00000060
1	/*	00000070
1	/*	00000080
1	/* HEADER DESCRIPTION	00000090
1	/*	00000100
1	/******	00000110
1		00000120
1	MODULE: MODEL_1;	00000130
2	SOURCE FILES: BANK,PARAM,INPUT,DOCUMENT;	00000140
3	TARGET FILES: REPORT,CONTROL,DOCUMENT;	00000150
4		00000160
4	/******	00000170
4	/*	00000180
4	/* FILES DESCRIPTION	00000190
4	/*	00000200
4	/******	00000210
4		00000220
4	DISK IS MEDIA (UNIT=3330);	00000230
5		00000240
5	/******	00000250
5	/*	00000260
5	/* DESCRIPTION OF PARAM FILE	00000270
5	/*	00000280
5	/******	00000290
5		00000300
5	PARAM IS FILE (DISK,SEQUENTIAL);	00000310
6	COEFF IS GROUP (PARAM);	00000320
7	CONSUMP IS RECORD (COEFF);	00000330
8	ALPHA IS FIELD (CONSUMP,DEC(14,8),(4));	00000340
9	INVESTM IS RECORD (COEFF);	00000350
10	BETA IS FIELD (INVESTM,DEC(14,8),(4));	00000360
11	WAGES IS RECORD (COEFF);	00000370
12	GAMMA IS FIELD (WAGES,DEC(14,8),(4));	00000380
13		00000390
13	/******	00000400
13	/*	00000410
13	/* DESCRIPTION OF BANK FILE	00000420
13	/*	00000430
13	/******	00000440
13		00000450
13	BANK IS FILE (DISK,KEY=NUMBER);	00000460
14	TIM_SER IS RECORD (BANK,(1));	00000470
15	NAME IS GROUP (TIM_SER);	00000480
16	LABEL IS FIELD (NAME,CHAR(4));	00000490
17	TITLE IS FIELD (NAME,CHAR(40));	00000500
18	NUMBER IS FIELD (TIM_SER,NUM(4));	00000510
19	TYPE IS GROUP (TIM_SER);	00000520
20	CODE IS FIELD (TYPE,CHAR(4));	00000530
21	UNITS IS FIELD (TYPE,CHAR(20));	00000540
22	RANGE IS GROUP (TIM_SER);	00000550
23	START IS GROUP (RANGE);	00000560

Figure A.1

```

24      YEAR IS FIELD (START,NUM(4));                                00000570
25      PERIOD IS FIELD (START,NUM(2));                              00000580
26      OBS_NUM IS FIELD (RANGE,NUM(3));                             00000590
27      NUM_PD_YR IS FIELD (RANGE,NUM(2));                           00000600
28      REF# IS FIELD (TIM_SER,NUM(3),(1:20));                       00000610
29      ENOREF IS FIELD (TIM_SER,CHAR(1));                            00000620
30      DATA IS FIELD (TIM_SER,DEC(10,5),(OBS_NUM));               00000630
31      /* BANK FILE IS DESCRIBED WITH A GENERAL FORMAT WHICH CAN  00000640
31      BE SHARED BY OTHER MODULES TO DESCRIBE THEIR DATA (EX.   00000650
31      ANNUAL AS WELL AS QUARTERLY DATA).                        00000660
31      FOR INSTANCE FOR KLEIN'S MODEL 1, WITH DATA FROM        00000670
31      1920 TO 1941, A TIME SERIES ENTRY WILL LOOK AS FOLLOWS:  00000680
31                                                                    00000690
31      (FIELD NAME)      (ENTRY)                                    00000700
31      NAME LABEL        C                                         00000710
31      NAME TITLE        CCNSUMPTION                               00000720
31      NUMBER            1                                           00000730
31      TYPE CODE         ENDO                                         00000740
31      TYPE UNITS        BILLIONS 1934 DOLLARS                      00000750
31      START YEAR        1920                                         00000760
31      START PERIOD      1                                           00000770
31      OBS_NUM           22                                           00000780
31      NUM_PD_YR         1                                           00000790
31      REF#(1)           1                                           00000800
31      REF#(2)           2                                           00000810
31      REF#(3)           3                                           00000820
31      ENOREF           8                                           00000830
31      DATA(1)          39.8                                         00000840
31      DATA(2)          41.9                                         00000850
31      .                                                         00000860
31      .                                                         00000870
31      .                                                         00000880
31      DATA(22)         69.7                                         00000890
31                                                                    00000900
31      /******//                                                  00000910
31      /*      DESCRIPTION OF DOCUMENT FILE                        /*/ 00000920
31      /*      /*/ 00000930
31      /*      /*/ 00000940
31      /******//                                                  00000950
31                                                                    00000960
31      DOCUMENT IS FILE (DISK,KEY=REF#);                             00000970
32      REFREC IS RECORD (DOCUMENT,(#));                              00000980
33      REF# IS FIELD (REFREC,NUM(3));                                00000990
34      DESCRIPTION IS GROUP (REFREC);                                00001000
35      TITLE IS FIELD (DESCRIPTION,CHAR(80));                        00001010
36      VOL# IS FIELD (DESCRIPTION,CHAR(4));                          00001020
37      DATE IS FIELD (DESCRIPTION,CHAR(13));                        00001030
38      PUBLISHER IS FIELD (DESCRIPTION,CHAR(20));                   00001040
39      AUTHOR IS FIELD (DESCRIPTION,CHAR(20));                      00001050
40      COMMENT IS FIELD (REFREC,CHAR(80));                           00001060
41                                                                    00001070
41      /******//                                                  00001080
41      /*      DESCRIPTION OF TARGET FILE CONTROL                /*/ 00001090
41      /*      /*/ 00001100
41      /******//                                                  00001110
41                                                                    00001120
41      CONTROL IS FILE (DISK,KEY=NUMBER);                            00001130
42      TIM_SER IS RECORD (CONTROL,(TCT_VAR));                       00001140
43                                                                    00001150
43      /*THE DESCRIPTION OF TIM_SER IS ASSUMED IDENTICAL TO THAT  00001160
43      PREVIOUSLY DESCRIBED FOR BANK.                               /*/ 00001170
43                                                                    00001180
43                                                                    00001190

```

A.1 (Cont.)

```

43  /*****
43  /*
43  /*      DESCRIPTION OF FILES IN TERMINAL      */
43  /*
43  /*****
43  /*****
43  /*
43  /*      DESCRIPTION OF SOURCE FILE INPUT      */
43  /*
43  /*****
43  /* INPUT CONTAINS CONTROL PARAMETERS FOR THE SIMULATION*/
43
43  INPUT IS FILE (TERM);
44  CNTRL_PARAM IS RECORD (INPUT);
45  SIM_YEAR IS FIELD (CNTRL_PARAM,NUM(4));
46  SET_PG IS FIELD (CNTRL_PARAM,NUM(2));
47  SIM_NP IS FIELD (CNTRL_PARAM,NUM(2));
48  SIM_TYPE IS FIELD (CNTRL_PARAM,CHAR(12));
49  LISTREP IS RECORD (INPUT);
50  TOT# IS FIELD (LISTREP,NUM(4));
51  VAR# IS FIELD (LISTREP,NUM(4),(TOT#));
52
52  /*****
52  /*
52  /*      DESCRIPTION OF TARGET FILE REPORT      */
52  /*
52  /*****
52
52  REPORT IS FILE (TERM);
53  OUTPUT IS GROUP (REPORT,(SIM_NP));
54  TITLE IS RECORD (OUTPUT);
55  YEAR IS FIELD (TITLE,PIC(4)A);
56  PERIOD IS FIELD (TITLE,PIC(6)A);
57  LABEL IS FIELD (TITLE,PIC(8)A);
58  VALUE IS FIELD (TITLE,PIC(8)A);
59  ENTRY IS RECORD (OUTPUT,(TOT#));
60  OUT_YEAR IS FIELD (ENTRY,PIC(4)Z);
61  OUT_PERIOD IS FIELD (ENTRY,PIC(6)Z);
62  OUT_LABEL IS FIELD (ENTRY,PIC(8)A);
63  OUT_VALUE IS FIELD (ENTRY,PIC(8)Z);
64
64  /* ASSERTIONS FOR TITLE HEADING */
64
64  YEAR = 'YEAR';
65  PERIOD = 'PERIOD';
66  LABEL = 'LABEL';
67  VALUE = 'VALUE';
68
68  /* THE OUTPUT REPORT FOR SELECTED VARIABLES WILL APPEAR
68  AS FOLLOWS IN THE TERMINAL: */
68  /*
68  YEAR__PERIOD__LABEL__VALUE
68  1921_____K_____184.121
68  /*
68  /* THE ENTRY RECORD WILL REPEAT FOR EVERY SELECTED VARIABLE
68  GIVEN IN INPUT.LISTREP, AND THE OUTPUT GROUP FOR EVERY PERIOD OF
68  SOLUTION. */
68
68  /* SINCE THE RECORDS IN THE SOURCE FILE 'BANK' ARE OF
68  (POSSIBLE) VARIABLE LENGTH, WITH (POSSIBLE) DIFFERENT
68  STARTING PERIODS AND NUMBER OF OBSERVATIONS (REFER TO

```

```

00001200
00001210
00001220
00001230
00001240
00001250
00001260
00001270
00001280
00001290
00001300
00001310
00001320
00001330
00001340
00001350
00001360
00001370
00001380
00001390
00001400
00001410
00001420
00001430
00001440
00001450
00001460
00001470
00001480
00001490
00001500
00001510
00001520
00001530
00001540
00001550
00001560
00001570
00001580
00001590
00001600
00001610
00001620
00001630
00001640
00001650
00001660
00001670
00001680
00001690
00001700
00001710
00001720
00001730
00001740
00001750
00001760
00001770
00001780
00001790
00001800
00001810
00001820
00001830

```

A.1 (Cont.)


```

68      FIGURE 3.41, IT BECOMES NECESSARY TO COMPUTE THE STARTING
68      PERIOD FOR THE SIMULATION (SOLUTION) AS AN OFFSET FROM
68      THE FIRST OBSERVATION ENTRY. */
68      /* NEXT ASSERTIONS COMPUTE THE OFFSET VALUE FOR EACH RECORD
68      AS WELL AS OTHER AUXILIARY PARAMETERS. */
68      /*****
68      /*
68      /*          INTERNAL PARAMETER DEFINITIONS          */
68      /*
68      /*****
68      OFFSET IS INTERIM (NUM(3),(TGT_VAR));
69      SAMPLE IS INTERIM (NUM(3),(TGT_VAR));
70      LAG IS INTERIM (NUM(3),(TGT_VAR));
71      NP IS SUBSCRIPT (CUTPUT,(1,SIM_NP,1,1));
72
72      /*****
72      /*          AUXILIARY ASSERTIONS FOR GENERAL SIMULATION          */
72      /*          SPECIFICATION          */
72      /*****
72      TGT_VAR = COUNT(BANK.TIM_SER); /* 'COUNT' FUNCTION RETURNS
73      TOTAL NUMBER OF RECORDS */
73      OFFSET(*) = (SIM_YEAR - START.YEAR(*) * NUM_PU_YR(*)
74      * (SIM_PD - START.PERIOD(*));
75      SAMPLE(*) = OFFSET(*) + NP;
76      MAXLAG = 1; /* LONGEST LAG EXPECTED FOR A MODEL */
77      I = NP + MAXLAG; /* 'TIME' INDEX FOR THE SOLUTION PERIOD */
78      NU = SIM_NP + MAXLAG; /* TOTAL NUMBER OF OBSERVATIONS */
79      LAG(*) = OFFSET(*) - MAXLAG;
79
79      /* IN ORDER TO FACILITATE THE TRANSCRIPTION OF RELATIONS,
79      INTERIM VARIABLES ARE USED TO DESCRIBE THE MODEL WITH
79      APPROPRIATE MNEMONICS. */
79      /*****
79      /*
79      /*          ENDOGENOUS          */
79      /*
79      /*****
79      C IS INTERIM (NUM(10),(INC)); /* CONSUMPTION */
80      I IS INTERIM (NUM(10),(INC)); /* NET INVESTMENT */
81      W1 IS INTERIM (NUM(10),(INC)); /* PRIVATE WAGE BILL */
82
82      /*****
82      /*
82      /*          DEFINITIONAL          */
82      /*
82      /*****
82      E IS INTERIM (NUM(10),(INC)); /* PRIVATE PRODUCT */
83      P IS INTERIM (NUM(10),(INC)); /* BUSINESS PROFITS */
84      W IS INTERIM (NUM(10),(INC)); /* TOTAL WAGE BILL */
85      K IS INTERIM (NUM(10),(INC)); /* CAPITAL STOCK
86      AT END OF YEAR */
86      Y IS INTERIM (NUM(10),(INC)); /* NATIONAL INCOME */
87
87      /*****
87      /*
87      /*          EXOGENOUS          */

```

A.1 (Cont.)


```

87      /*                                                    */ 00002470
87      /******                                                    */ 00002480
87      /******                                                    */ 00002490
87      G IS INTERIM (INOM(10), (NG)); /* GOVERNMENT SPENDING */ 00002500
88      R IS INTERIM (INOM(10), (NG)); /* BUSINESS TAXES */ 00002510
89      T1 IS INTERIM (INOM(10), (NG)); /* TREND VARIABLE */ 00002520
90      W2 IS INTERIM (INOM(10), (NG)); /* GOVERNMENTAL WAGE 00002530
91      BILL */ 00002540
91      /* ASSIGNMENT OF EXOGENOUS VALUES TO INTERIM VARIABLES */ 00002550
91      /* THIS PARTICULAR REPRESENTATION REQUIRES THE USER TO */ 00002560
91      /* KNOW THE PHYSICAL LOCATION OF DATA (TIME-SERIES) IN */ 00002570
91      /* THE 'BANK' FILE. */ 00002580
91      /******                                                    */ 00002590
91      VAR#1: G(T) = BANK.DAT(1, SAMPLE(1)); 00002600
91      VAR#2: R(T) = BANK.DAT(2, SAMPLE(2)); 00002610
92      VAR#3: T1(T) = BANK.DAT(3, SAMPLE(3)); 00002620
93      VAR#4: W2(T) = BANK.DAT(4, SAMPLE(4)); 00002630
94      VAR#5: W1(T) = BANK.DAT(5, SAMPLE(5)); 00002640
95      /* OTHER ENDOGENOUS AND DEFINITIONAL VARIABLES ARE */ 00002650
95      /* ASSIGNED AS FOLLOWS: */ 00002660
95      /******                                                    */ 00002670
95      /* G TO VAR#5 */ 00002680
95      /* 1 TO VAR#6 */ 00002690
95      /* W1 TO VAR#7 */ 00002700
95      /* E TO VAR#8 */ 00002710
95      /* P TO VAR#9 */ 00002720
95      /* K TO VAR#10 */ 00002730
95      /* K TO VAR#11 */ 00002740
95      /* Y TO VAR#12 */ 00002750
95      /******                                                    */ 00002760
95      /* ASSIGNMENT OF LAGGED HISTORICAL DATA TO */ 00002770
95      /* INTERIM VARIABLES */ 00002780
95      /******                                                    */ 00002790
95      FOR J = 1 TO MAXLAG LET E(J) = BANK.DAT(8, LAG(8)+J); 00002800
95      FOR J = 1 TO MAXLAG LET P(J) = BANK.DAT(9, LAG(9)+J); 00002810
96      FOR J = 1 TO MAXLAG LET K(J) = BANK.DAT(11, LAG(11)+J); 00002820
97      /******                                                    */ 00002830
97      /* MODEL 1 EQUATIONS */ 00002840
97      /******                                                    */ 00002850
97      /******                                                    */ 00002860
97      /******                                                    */ 00002870
97      /******                                                    */ 00002880
97      INCOME: 00002890
98      Y(T) = G(T) + T1(T) + G(T) - R(T); 00002900
99      PRIV_PROD: 00002910
99      E(T) = Y(T) + R(T) - W2(T); 00002920
100      T1_WAGE: 00002930
100      W1(T) = W1(T) + W2(T); 00002940
101      PROFITS: 00002950
101      P(T) = Y(T) - W(T); 00002960
102      CONSUMPTION: 00002970
102      C(T) = ALPHA(1) + ALPHA(2) * W(T) 00002980
102      + ALPHA(3) * P(T) + ALPHA(4) * P(T-1); 00002990
103      INVESTMENT: 00003000
103      I(T) = BETA(1) + BETA(2) * P(T) 00003010
103      + BETA(3) * P(T-1) + BETA(4) * K(T-1); 00003020
104      PRIV_WAGE: 00003030
104      W1(T) = GAMMA(1) + GAMMA(2) * E(T) 00003040
104      + GAMMA(3) * E(T-1) + GAMMA(4) * (I(T)-I(T-1)); 00003050
105      CAPITAL: 00003060
105      K(T) = I(T) + K(T-1); 00003070
106      00003080
106      00003090

```

A.1 (Cont.)

```

106 /*****
106 /*
106 /*          ASSERTIONS FOR FILE REPORT          */
106 /*
106 /*****
106 N IS SUBSCRIPT (ENTRY, (1, TCT#, 1, 1));
107 OUT_YEAR(N,NP) = SIM_YEAR + (NP + SIM_PD - 2)/NUM_PD_YR;
108 OUT_PERIOD(N,NP) = MOD (NP + SIM_PD - 2, NUM_PD_YR) + 1;
109 OUT_LABEL(N,NP) = BANK.LABEL(VAR#(N));
110 OUT_VALUE(N,NP) = CONTROL.DATA(VAR#(N),T(NP));
111
111 /*****
111 /*
111 /*          ASSERTIONS FOR FILE CONTROL          */
111 /*
111 /*****
111 CONTROL.START.YEAR(*) = SIM_YEAR;
112 CONTROL.START.PERIOD(*) = SIM_PD;
113 CONTROL.OBS_NUM(*) = SIM_NP;
114 CONTROL.DATA(1,*) = G(T);
115 CONTROL.DATA(2,*) = R(T);
116 CONTROL.DATA(3,*) = I1(T);
117 CONTROL.DATA(4,*) = W2(T);
118 CONTROL.DATA(5,*) = C(T);
119 CONTROL.DATA(6,*) = I(T);
120 CONTROL.DATA(7,*) = W1(T);
121 CONTROL.DATA(8,*) = E(T);
122 CONTROL.DATA(9,*) = P(T);
123 CONTROL.DATA(10,*) = W(T);
124 CONTROL.DATA(11,*) = K(T);
125 CONTROL.DATA(12,*) = Y(T);
126 /*****
126 /*
126 /*          INTERFILE RELATIONSHIPS          */
126 /*
126 /*****
126 PGINTER.CONTROL.TIM_SER = BANK.NUMBER;
127 PUNTER.DOCUMENT.REFREC = CONTROL.REF#(*);
128
128 /* ASSERTION TO COMPUTE THE NUMBER OF REPETITIONS OF FIELD
128 REF# */
128 EXIST.BANK.REF# = DELIMITIM_SER,'#')/3;
129 /*DELIM' FUNCTION SCANS FOR '#' DELIMITER SYMBOL IN THE
129 RECORD TIM_SER (FIELD ENREF) */
129
129 FOR J = 1 TO 4 LET
129 EXIST.CONTROL.REF#(J) = EXIST.BANK.REF#(J);
129 FOR J = 5 TO 12 LET EXIST.CONTROL.REF#(J) = 1;
130 FOR J = 5 TO 12 LET CONTROL.REF#(J,1) = 99;
131 /* RECORD 99 IN FILE DOCUMENT WILL BE SET TO INDICATE THAT
131 ENDogenous VARIABLES ARE GENERATED FROM THIS PARTICULAR
131 SIMULATION RUN */
131 DOCUMENT.REFREC.REF#(99) = 99;
133 DOCUMENT.REFREC.TITLE(99) = 'MODEL_1 DYNAMIC SIMULATION';
134 DOCUMENT.REFREC.VOL#(99) = 'VOLUME';
135 DOCUMENT.REFREC.DATE(99) = 'NOV-4-77';
136 DOCUMENT.REFREC.PUBLISHER(99) = 'VOLUME';
137 DOCUMENT.REFREC.AUTHOR(99) = 'L.R.KLEIN';
138 DOCUMENT.REFREC.COMMENT(99) = 'CONTROL SOLUTION GENERATED
138 FROM SPECIFICATION IN MODEL LANGUAGE'

```

A.1 (Cont.)

```

138 /*****
138 /*
138 /* FOLLOWING STATEMENTS WERE GENERATED BY THE SYSTEM
138 /*
138 /*****
139 TOT_VAR IS INTERIM;
140 NO IS INTERIM;
141 TERM IS MEDIA;
142 NAME IS GROUP(TIM_SER);
143 NUMBER IS FIELD(TIM_SER,NUMERIC(4));
144 TYPE IS GROUP(TIM_SER);
145 RANGE IS GROUP(TIM_SER);
146 REF# IS FIELD(TIM_SER,(20),NUMERIC(3));
147 ENDREF IS FIELD(TIM_SER,CHAR(1));
148 DATA IS FIELD(TIM_SER,DOCS_NUM),DECIMAL(10,5));
149 START IS GROUP(RANGE);
150 GDS_NUM IS FIELD(RANGE,NUMERIC(3));
151 NUM_PD_YR IS FIELD(RANGE,NUMERIC(2));
152 CODE IS FIELD(TYPE,CHAR(4));
153 UNITS IS FIELD(TYPE,CHAR(20));
154 LABEL IS FIELD(NAME,CHAR(4));
155 TITLE IS FIELD(NAME,CHAR(40));
156 YEAR IS FIELD(START,NUMERIC(4));
157 PERIOD IS FIELD(START,NUMERIC(2));
158 EXIST IS GROUP;
159 DISK IS GROUP(EXIST);
160 DOCUMENT IS GROUP(DISK);
161 REFREC IS INTERIM(DOCUMENT);
162 BANK IS GROUP(DISK);
163 TIM_SER IS INTERIM(BANK);
164 J IS SUBSCRIPT(1,*,1,1));
165 CONTROL IS GROUP(EXIST);
166 REF# IS FIELD(CONTROL);
167 REF# IS FIELD(BANK);
168 PCINTER IS GROUP;
169 DOCUMENT IS GROUP(PCINTER);
170 REFREC IS FIELD(DOCUMENT);
171 CONTROL IS GROUP(PCINTER);
172 TIM_SER IS FIELD(CONTROL);
173 F IS INTERIM;
174 $A$10001: $10001=VAR#(N);
175 $A$10002: $10002=T(INP);
176 $10001 IS INTERIM;
177 $10002 IS INTERIM;
178 $A$10004: $10004=SIM_PD - 2;
179 $10004 IS INTERIM;
180 $A$10005: $10005=T - 1;
181 $10005 IS INTERIM;
182 $A$10010: $10010=LAG(11);
183 $A$10010 IS INTERIM;
184 $10010 IS INTERIM;
185 $A$10011: $10011=LAG(9);
186 $10011 IS INTERIM;
187 $A$10012: $10012=LAG(2);
188 $10012 IS INTERIM;
189 $A$10013: $10013=SAMPLE(4);
190 $10013 IS INTERIM;
191 $A$10014: $10014=SAMPLE(3);
192 $10014 IS INTERIM;
193 $A$10015: $10015=SAMPLE(2);
194 $10015 IS INTERIM;
195 $A$10016: $10016=SAMPLE(1);
196 $10016 IS INTERIM;

```

A.1 (Cont)

AD-A066 192

MOORE SCHOOL OF ELECTRICAL ENGINEERING PHILADELPHIA P--ETC F/G 9/2
USE AND EXTENSION OF AN AUTOMATIC PROGRAM GENERATOR FOR MODEL B--ETC(U)
OCT 78 J L GANA

N00014-76-C-0416

UNCLASSIFIED

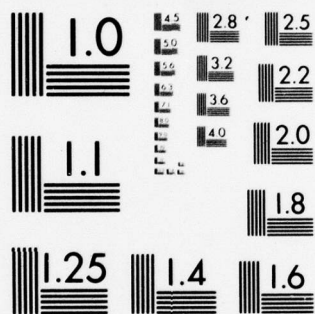
78-02

NL

5 OF 6

AD
A088102





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

197	TERM IS MEDIA;	*****
198	DISK IS MEDIA(UNIT=3330);	*****
199	\$B1001 IS SUBSCRIPT(1,*,1,1);	*****
200	\$B1002 IS SUBSCRIPT(1,*,1,1);	*****
201	\$B1003 IS SUBSCRIPT(1,*,1,1);	*****
202	\$B1004 IS SUBSCRIPT(1,*,1,1);	*****
203	\$B1005 IS SUBSCRIPT(1,*,1,1);	*****
204	\$B1006 IS SUBSCRIPT(1,*,1,1);	*****
205	\$B1007 IS SUBSCRIPT(1,*,1,1);	*****
206	\$B1008 IS SUBSCRIPT(1,*,1,1);	*****

1 THE FOLLOWING GROUPS OF ITEMS ARE CIRCULARLY DESCRIBED
 (DICT# , PREFIX CODE , NAME)

C

196	\$I	E
259	\$I	E
117	\$A	PRIV_PROD
200	\$I	I
246	\$I	I
113	\$A	INVESTMENT
194	\$I	P
257	\$I	P
115	\$A	PROFITS
186	\$I	F
260	\$I	Y
118	\$A	INCOME
202	\$I	C
252	\$I	C
114	\$A	CONSUMPTION
192	\$I	M
258	\$I	M
116	\$A	TOT_WAGE
198	\$I	W1
240	\$I	W1
112	\$A	PRIV_WAGE

Figure A.2

```

/*****
/*
/*          MODEL_1 MODULE DESCRIPTION
/*
/*
/*****
MODULE: MODEL_1;
SOURCE FILES: INPUT,DECMNT,BANK,PARAM;
TARGET FILES: REPORT,DECMNT;

/*****
/*
/*          DATA DESCRIPTION STATEMENTS
/*
/*
/*****
DISK IS MEDIA(UNIT=3330);
TERM IS MEDIA;
TERM IS MEDIA;
DISK IS MEDIA(UNIT=3330);

/*****
/*
/*          "REPORT" FILE DESCRIPTION
/*
/*
/*
/*****
REPORT IS FILE(TERM);
OUTPUT IS GROUP(REPORT,(SIM_NP));
TITLE IS RECORD(OUTPUT);
YEAR IS FIELD(TITLE,PICTURE '(4)A');
PERIOD IS FIELD(TITLE,PICTURE 'B(6)A');
LABEL IS FIELD(TITLE,PICTURE 'C(5)A');
VALUE IS FIELD(TITLE,PICTURE 'E(5)A');
ENTRY IS RECORD(OUTPUT,(TOT));
CUT_YEAR IS FIELD(ENTRY,PICTURE 'ZZZ9');
C IS FIELD(ENTRY,PICTURE '(6)Z9');
C IS FIELD(ENTRY,PICTURE 'E(4)A');
CUT_VALUE IS FIELD(ENTRY,PICTURE 'E(5ZZ9V999');

/*****
/*
/*          "INPUT" FILE DESCRIPTION
/*
/*
/*
/*****
INPUT IS FILE(TERM);
CNTRL_PARAM IS RECORD(INPUT);
SIM_YEAR IS FIELD(CNTRL_PARAM,NUMERIC(4));
SIM_PJ IS FIELD(CNTRL_PARAM,NUMERIC(2));
SIM_NP IS FIELD(CNTRL_PARAM,NUMERIC(2));
SIM_TYPE IS FIELD(CNTRL_PARAM,CHAR(12));
LISTREP IS RECORD(INPUT);
TCT# IS FIELD(LISTREP,NUMERIC(4));
VAR# IS FIELD(LISTREP,(TCT#),NUMERIC(4));

/*****
/*
/*          "COUTFUL" FILE DESCRIPTION
/*
/*
/*
/*****

```

Figure A.3


```

CONTROL IS FILE(DISK,KEY=NUMBER);                                00002600
TIM_SER IS RECORD(CONTROL,(TGT_VAR));                            00002700
NAME IS GROUP(TIM_SER);                                          00002800
LABEL IS INTERIMNAME,CHAR(4));                                  00002900
TITLE IS INTERIMNAME,CHAR(40));                                  00003000
NUMBER IS INTERIM(TIM_SER,NUMERIC(4));                           00003100
TYPE IS GROUP(TIM_SER);                                          00003200
CODE IS INTERIMTYPE,CHAR(4));                                    00003300
UNITS IS INTERIMTYPE,CHAR(20));                                  00003400
RANGE IS GROUP(TIM_SER);                                         00003500
START IS GROUP(RANGE);                                           00003600
YEAR IS INTERIMSTART,NUMERIC(4));                                00003700
PERIOD IS INTERIMSTART,NUMERIC(2));                              00003800
CDS_NUM IS INTERIMRANGE,NUMERIC(3));                             00003900
NUM_PO_YR IS INTERIMRANGE,NUMERIC(2));                           00004000
REF# IS INTERIM(TIM_SER,(20),NUMERIC(3));                       00004100
ENDREF IS INTERIM(TIM_SER,CHAR(1));                               00004200
DATA IS INTERIM(TIM_SER,(CDS_NUM),DECIMAL(10,5));               00004300
                                                                    00004301
/*****                                                             00004302
/*                                                                    */ 00004303
/*          "JOCMENT" FILE DESCRIPTION                             */ 00004304
/*                                                                    */ 00004305
/*****                                                             00004306
JOCMENT IS FILE(DISK,KEY=REF#);                                  00004307
REFREC IS RECORD(JOCMENT,(EXIST.DISK.JOCMENT,REFREC));          00004400
REF# IS FIELD(REFREC,NUMERIC(3));                                00004500
DESCRIPTION IS GROUP(REFREC);                                    00004600
TITLE IS FIELD(DESCRIPTION,CHAR(80));                            00004700
VOL# IS FIELD(DESCRIPTION,CHAR(4));                              00004800
DATE IS FIELD(DESCRIPTION,CHAR(13));                             00004900
PUBLISHER IS FIELD(DESCRIPTION,CHAR(20));                        00005000
AUTHOR IS FIELD(DESCRIPTION,CHAR(20));                           00005100
COMMENT IS FIELD(REFREC,CHAR(80));                               00005200
                                                                    00005300
/*****                                                             00005301
/*                                                                    */ 00005302
/*          "BANK" FILE DESCRIPTION                             */ 00005303
/*                                                                    */ 00005304
/*                                                                    */ 00005305
/*****                                                             00005306
BANK IS FILE(DISK,KEY=NUMBER);                                  00005307
TIM_SER IS RECORD(BANK,(EXIST.DISK.BANK.TIM_SER));              00005400
NAME IS GROUP(TIM_SER);                                          00005500
LABEL IS FIELD(NAME,CHAR(4));                                    00005600
TITLE IS FIELD(NAME,CHAR(40));                                   00005700
NUMBER IS FIELD(TIM_SER,NUMERIC(4));                              00005800
TYPE IS GROUP(TIM_SER);                                          00005900
CODE IS FIELD(TYPE,CHAR(4));                                     00006000
UNITS IS FIELD(TYPE,CHAR(20));                                   00006100
RANGE IS GROUP(TIM_SER);                                         00006200
START IS GROUP(RANGE);                                           00006300
YEAR IS FIELD(START,NUMERIC(4));                                 00006400
PERIOD IS FIELD(START,NUMERIC(2));                               00006500
CDS_NUM IS FIELD(RANGE,NUMERIC(3));                              00006600
NUM_PO_YR IS FIELD(RANGE,NUMERIC(2));                           00006700
REF# IS FIELD(TIM_SER,(20),NUMERIC(3));                         00006800
ENDREF IS FIELD(TIM_SER,CHAR(1));                                00006900
DATA IS FIELD(TIM_SER,(CDS_NUM),DECIMAL(10,5));                00007000
                                                                    00007100
/*****                                                             00007101
/*                                                                    */ 00007102
/*                                                                    */ 00007103

```

A.3 (Cont.)

```

/*          "PARAM" FILE DESCRIPTION          */ 00007104
/*          */ 00007105
/*          */ 00007106
/*          */ 00007107
PARAM IS FILE(DISK,SAM); 00007200
COEFF IS GROUP(PARAM); 00007300
CONSUMP IS RECCO(COEFF); 00007400
ALPHA IS FIELD(CONSUMP,(4),DECIMAL(14,8)); 00007500
INVESTM IS RECCO(COEFF); 00007600
BETA IS FIELD(INVESTM,(4),DECIMAL(14,8)); 00007700
WAGES IS RECCO(COEFF); 00007800
GAMMA IS FIELD(WAGES,(4),DECIMAL(14,8)); 00007900
/*          */ 00007901
/*          */ 00007902
/*          */ 00007903
/*          INTERIM DATA DESCRIPTION STATEMENTS  */ 00007904
/*          */ 00007905
/*          */ 00007906
/*          */ 00007907
PCINTER IS GROUP; 00008000
CONTROL IS GROUP(PCINTER); 00008100
TIM_SER IS INTERIM(CONTROL,(*)); 00008200
DECCMENT IS GROUP(PCINTER); 00008300
REFREC IS INTERIM(DECCMENT,(*,*)); 00008400
EXIST IS GROUP; 00008500
CONTROL IS GROUP(EXIST); 00008600
REF# IS INTERIM(CONTROL,(*)); 00008700
DISK IS GROUP(EXIST); 00008800
BANK IS GROUP(DISK); 00008900
REF# IS INTERIM(BANK,(*)); 00009000
TIM_SER IS INTERIM(BANK); 00009100
DECCMENT IS GROUP(DISK); 00009200
REFREC IS INTERIM(DECCMENT); 00009300
$10016 IS INTERIM(*); 00009400
$10015 IS INTERIM(*); 00009500
$10014 IS INTERIM(*); 00009600
$10013 IS INTERIM(*); 00009700
$10012 IS INTERIM; 00009800
$10011 IS INTERIM; 00009900
$10010 IS INTERIM; 00010000
MAXLAG IS INTERIM; 00010100
$10005 IS INTERIM(*); 00010200
$10004 IS INTERIM; 00010300
$10002 IS INTERIM(*); 00010400
$10001 IS INTERIM(*); 00010500
T IS INTERIM(*); 00010600
NC IS FIELD; 00010700
TCT_VAR IS FIELD; 00010800
W2 IS INTERIM(*,NC),NUMERIC(10); 00010900
T1 IS INTERIM(*,NC),NUMERIC(10); 00011000
R IS INTERIM(*,NC),NUMERIC(10); 00011100
G IS INTERIM(*,NC),NUMERIC(10); 00011200
Y IS INTERIM(*,NC),NUMERIC(10); 00011300
K IS INTERIM(*,NC),NUMERIC(10); 00011400
W IS INTERIM(*,NC),NUMERIC(10); 00011500
P IS INTERIM(*,NC),NUMERIC(10); 00011600
E IS INTERIM(*,NC),NUMERIC(10); 00011700
W1 IS INTERIM(*,NC),NUMERIC(10); 00011800
I IS INTERIM(*,NC),NUMERIC(10); 00011900
C IS INTERIM(*,NC),NUMERIC(10); 00012000
LAG IS INTERIM(TCT_VAR),NUMERIC(3); 00012100
SAMPLE IS INTERIM(*,TCT_VAR),NUMERIC(3); 00012200
OFFSET IS INTERIM(TCT_VAR),NUMERIC(3); 00012300

```

```

/*****
/*
/*          SUBSCRIPT DESCRIPTION STATEMENTS
/*
/*
/*****
00C12301
00C12302
00C12303
00C12304
00C12305
00C12306
00C12307
00C12400
00C12500
00C12600
00C12700
00C12800
00C12900
00C13000
00C13001
00C13002
00C13003
00C13004
00C13005
00C13006
00C13007
00C13008
00C13009
00C13010
00C13011
00C13012
00C13013
00C13014
00C13015
00C13016
00C13017
00C13018
00C13019
00C13020
00C13021
00C13022
00C13023
00C13024
00C13025
00C13026
00C13027
00C13028
00C13029
00C13030
00C13031
00C13032
00C13033
00C13034
00C13035
00C13036
00C13037
00C13038
00C13039
00C13040
00C13041
00C13042
00C13043
00C13044
00C13045
00C13046
00C13047
00C13048
00C13049

ForeachLAG IS SUBSCRIPT(LAG,(1,TOT_VAR,1,1));
ForeachTIM_SER IS SUBSCRIPT(TIM_SER,(1,TOT_VAR,1,1));
ForeachREF# IS SUBSCRIPT(REF#,(1,C6,1,1));
ForeachREF# IS SUBSCRIPT(REF#,(1,C6,1,1));
J IS SUBSCRIPT(JDATA,(1,CBS_NCM,1,1));
N IS SUBSCRIPT(OUTPCT,(1,SIM_NP,1,1));
NP IS SUBSCRIPT(ENTRY,(1,TOT_N,1,1));

/*****
/*
/*          ASSERTIONS
/*
/*
/*****
00C13002
00C13003
00C13004
00C13005
00C13006
00C13007
00C13008
00C13009
00C13010
00C13011
00C13012
00C13013
00C13014
00C13015
00C13016
00C13017
00C13018
00C13019
00C13020
00C13021
00C13022
00C13023
00C13024
00C13025
00C13026
00C13027
00C13028
00C13029
00C13030
00C13031
00C13032
00C13033
00C13034
00C13035
00C13036
00C13037
00C13038
00C13039
00C13040
00C13041
00C13042
00C13043
00C13044
00C13045
00C13046
00C13047
00C13048
00C13049

$A$10016:
$10016(NP)=SAMPLE(NP,1);

$A$10015:
$10015(NP)=SAMPLE(NP,2);

$A$10014:
$10014(NP)=SAMPLE(NP,3);

$A$10013:
$10013(NP)=SAMPLE(NP,4);

$A$10012:
$10012=LAG(8);

$A$10011:
$10011=LAG(9);

$A$10010:
$10010=LAG(11);

$A$10005:
$10005(NP)=T(NP) - 1;

$A$10004:
$10004=SIM_PD - 2;

$A$10002:
$10002(NP)=T(NP);

$A$10001:
$10001(N)=VAL#(N);

$A0046:
TARGET.COMMENT(99)='CONTROL SOLUTION GENERATED
FROM SPECIFICATION IN MODEL LANGUAGE';

$A0045:
TARGET.AUTHOR(99)='L.R.KLEIN';

$A0044:

```

A.3 (Cont.)

TARGET.PUBLISHER(99)='VOID';	00013050
\$A0043:	00013051
TARGET.DATE(99)='NCV-4-77';	00013052
	00013053
	00013054
\$A0042:	00013055
TARGET.VCL#(99)='VCIC';	00013056
	00013057
\$A0041:	00013058
TARGET.DESCRPTION.TITLE(99)='MODEL_1 DYNAMIC SIMULATION';	00013059
	00013060
\$A0040:	00013061
TARGET.DISK.DOCUMENT.REFREC.REF#(99)=99;	00013062
	00013063
\$A0039:	00013064
FOR J = 5 TO 12 DISK.CONTROL.TIM_SER.REF#(J,1)=99;	00013065
	00013066
\$A0038:	00013067
FOR J = 5 TO 12 EXIST.CONTROL.REF#(J)=1;	00013068
	00013069
\$A0037:	00013070
FOR J = 1 TO 4 EXIST.CONTROL.REF#(J)=EXIST.BANK.REF#(J);	00013071
	00013072
\$A0036:	00013073
EXIST.BANK.REF#(FOREACH\$REF#)=DELIM(EXIST.BANK.TIM_SER,'#') / 3;	00013074
	00013075
\$A0035:	00013076
PCINTER.REFREC(FOREACH\$REF#,FOREACH\$TIM_SER)=	00013077
DISK.CONTROL.TIM_SER.REF#(FOREACH\$TIM_SER,FOREACH\$REF#);	00013078
	00013079
\$A0034:	00013080
PCINTER.CONTROL.TIM_SER(N)=DISK.BANK.TIM_SER.NUMBER(N);	00013081
	00013082
\$A0033:	00013083
DISK.CONTROL.TIM_SER.DATA(12,NP)=Y(NP,T(NP));	00013084
	00013085
\$A0032:	00013086
DISK.CONTROL.TIM_SER.DATA(11,NP)=X(NP,T(NP));	00013087
	00013088
\$A0031:	00013089
DISK.CONTROL.TIM_SER.DATA(10,NP)=W(NP,T(NP));	00013090
	00013091
\$A0030:	00013092
DISK.CONTROL.TIM_SER.DATA(9,NP)=P(NP,T(NP));	00013093
	00013094
\$A0029:	00013095
DISK.CONTROL.TIM_SER.DATA(8,NP)=E(NP,T(NP));	00013096
	00013097
\$A0028:	00013098
DISK.CONTROL.TIM_SER.DATA(7,NP)=W1(NP,T(NP));	00013099
	00013100
\$A0027:	00013101
DISK.CONTROL.TIM_SER.DATA(6,NP)=I(NP,T(NP));	00013102
	00013103
\$A0026:	00013104
DISK.CONTROL.TIM_SER.DATA(5,NP)=C(NP,T(NP));	00013105
	00013106
\$A0025:	00013107
DISK.CONTROL.TIM_SER.DATA(4,NP)=W2(NP,T(NP));	00013108
	00013109
\$A0024:	00013110
DISK.CONTROL.TIM_SER.DATA(3,NP)=T1(NP,T(NP));	00013111
	00013112

A.3 (Cont.)

\$ACC23:	DISK.CCCTRL.TIM_SER.DATA(2,NP)=K(NP,T(NP));	00013113
		00013114
\$ACC22:	DISK.CCCTRL.TIM_SER.DATA(1,NP)=G(NP,T(NP));	00013115
		00013116
\$ACC21:	DISK.CCCTRL.TIM_SER.RANGE.GES_NUM(FOR EACH \$TIM_SER)=SIM_NP;	00013117
		00013118
		00013119
		00013120
\$ACC20:	DISK.CCCTRL.TIM_SER.RANGE.START.PERIOD(FOR EACH \$TIM_SER)=SIM_PD;	00013121
		00013122
		00013123
\$ACC19:	DISK.CCCTRL.TIM_SER.RANGE.START.YEAR(FOR EACH \$TIM_SER)=SIM_YEAR;	00013124
		00013125
		00013126
CAPITAL:	K(NP,T(NP))=E(NP,T(NP)) * K(NP,SIGC05(NP));	00013127
		00013128
\$ACC14:	FOR J = 1 TO MAXLAG K(NP,J)=DISK.BANK.TIM_SER.DATA(11,\$IGJ10 + J);	00013129
		00013130
		00013131
		00013132
		00013133
\$ACC13:	FOR J = 1 TO MAXLAG P(NP,J)=DISK.BANK.TIM_SER.DATA(9,\$IGJ11 + J);	00013134
		00013135
		00013136
		00013137
\$ACC12:	FOR J = 1 TO MAXLAG E(NP,J)=DISK.BANK.TIM_SER.DATA(8,\$IGJ12 + J);	00013138
		00013139
		00013140
		00013141
VAR#4:	W(NP,T(NP))=DISK.BANK.TIM_SER.DATA(14,\$IGC13(NP));	00013142
		00013143
VAR#3:	T(NP,T(NP))=DISK.BANK.TIM_SER.DATA(3,\$IGC14(NP));	00013144
		00013145
VAR#2:	R(NP,T(NP))=DISK.BANK.TIM_SER.DATA(2,\$IGC15(NP));	00013146
		00013147
VAR#1:	G(NP,T(NP))=DISK.BANK.TIM_SER.DATA(1,\$IGC16(NP));	00013148
		00013149
		00013150
		00013151
		00013152
\$ACC11:	LAG(FOR EACH \$LAG)=OFFSET(FOR EACH \$LAG) - MAXLAG;	00013153
		00013154
\$ACC09:	T(NP)=NP + MAXLAG;	00013155
		00013156
		00013157
		00013158
		00013159
\$ACC08:	MAXLAG=1;	00013160
		00013161
\$ACC07:	SAMPLE(NP,FOR EACH \$LAG)=OFFSET(FOR EACH \$LAG) * NP;	00013162
		00013163
		00013164
\$ACC06:	OFFSET(N)=SIM_YEAR - DISK.BANK.TIM_SER.RANGE.START.YEAR(N) * DISK.BANK.TIM_SER.RANGE.NUM_PD_YEAR(N) + (SIM_PD - DISK.BANK.TIM_SER.RANGE.START.PERIOD(N));	00013165
		00013166
		00013167
		00013168
		00013169
		00013170
\$ACC01:	OUTPUT.YEAR(N)='YEAR';	00013171
		00013172
\$ACC02:		00013173
		00013174
		00013175

A.3 (Cont.)

```

OUTPUT.PERIOD(N)=*PERIOD*;                                00013176
SACC03:                                                    00013177
  OUTPUT.LABEL(N)='LABEL';                                00013178
SACC04:                                                    00013179
  VALUE(N)='VALUE';                                       00013180
SACC15:                                                    00013181
  OUT_YEAR(N,NP)=SIM_YEAR + (NP + SIM_PD - 2) /         00013182
  DISK.BANK.TIM_SER.RANGE.NUM_PD_YR(N);                 00013183
SACC16:                                                    00013184
  O(N,NP)=MOD(NP + $10004,DISK.BANK.TIM_SER.RANGE.NUM_PD_YR(N)) + 1;00013185
SACC17:                                                    00013186
  C(N,NP)=DISK.BANK.TIM_SER.NAME.LABEL($10001(N));       00013187
SACC18:                                                    00013188
  OUT_VALUE(N,NP)=DISK.CONTROL.TIM_SER.DATA($10001(N),$10002(NP)); 00013189
SACC05:                                                    00013190
  TCT_VAR=COUNT(EXIST.BANK.TIM_SER);                    00013191
SACC10:                                                    00013192
  NC=SIM_NP + MAXLAG;                                    00013193
/******                                                    00013194
/*                                                    00013195
/*              SIMULTANEOUS ASSERTIONS                    00013196
/*                                                    00013197
/******                                                    00013198
PRIV_PROD:                                                    00013199
  E(NP,T(NP))=Y(NP,T(NP)) + K(NP,T(NP)) - W2(NP,T(NP)); 00013200
INVESTMENT:                                                    00013201
  I(NP,T(NP))=JETA(1) + BETA(2) * P(NP,T(NP)) + BETA(3) * P(NP, 00013202
  $10005(NP)) + BETA(4) * K(NP,$10005(NP));              00013203
PROFITS:                                                    00013204
  P(NP,T(NP))=Y(NP,T(NP)) - W(NP,T(NP));                 00013205
INCOME:                                                    00013206
  Y(NP,T(NP))=C(NP,T(NP)) + I(NP,T(NP)) + G(NP,T(NP)) - K(NP,T(NP)) 00013207
;                                                            00013208
CONSUMPTION:                                                    00013209
  C(NP,T(NP))=ALPHA(1) + ALPHA(2) * W(NP,T(NP)) + ALPHA(3) * P(NP, 00013210
  T(NP)) + ALPHA(4) * P(NP,$10005(NP));                   00013211
TOT_WAGE:                                                    00013212
  W(NP,T(NP))=W1(NP,T(NP)) + W2(NP,T(NP));               00013213
PRIV_WAGE:                                                    00013214
  W1(NP,T(NP))=GAMMA(1) + GAMMA(2) * E(NP,T(NP)) + GAMMA(3) * E( 00013215
  NP,$10005(NP)) + GAMMA(4) * (T(NP,T(NP)) - 1935);      00013216
/******                                                    00013217
/*                                                    00013218
/*              END OF SIMULTANEOUS GROUP                    00013219
/*                                                    00013220
/******                                                    00013221

```

A.3 (Cont.)

ATTRIBUTES AND CROSS-REFERENCE TABLE
ATTITUDE AND REFERENCES

SPMT NO.	QUL NO.	QULCT NO.	IDENTIFICATION	ATTRIBUTE AND REFERENCES
174	000000	217	SAS10001	ASSERTION NAME
175	000000	218	SAS10002	ASSERTION NAME
176	000000	219	SAS10003	ASSERTION NAME
180	000000	225	SAS10005	ASSERTION NAME
182	000000	261	SAS10010	ASSERTION NAME
185	000000	400	SAS10011	ASSERTION NAME
187	000000	415	SAS10012	ASSERTION NAME
189	000000	276	SAS10013	ASSERTION NAME
191	000000	203	SAS10014	ASSERTION NAME
193	000000	280	SAS10015	ASSERTION NAME
195	000000	295	SAS10016	ASSERTION NAME
64	1070	130	SAC001	ASSERTION NAME
65	1080	132	SAC002	ASSERTION NAME
66	1070	134	SAC003	ASSERTION NAME
67	1080	132	SAC004	ASSERTION NAME
72	2070	132	SAC005	ASSERTION NAME
73	2100	131	SAC006	ASSERTION NAME
74	2101	130	SAC007	ASSERTION NAME
75	2120	129	SAC008	ASSERTION NAME
76	2121	129	SAC009	ASSERTION NAME
77	2140	127	SAC010	ASSERTION NAME
78	2150	126	SAC011	ASSERTION NAME
55	2020	121	SAC012	ASSERTION NAME
56	2030	120	SAC013	ASSERTION NAME
57	2040	119	SAC014	ASSERTION NAME
107	3152	110	SAC015	ASSERTION NAME
108	3172	109	SAC016	ASSERTION NAME
109	3190	108	SAC017	ASSERTION NAME
110	3200	107	SAC018	ASSERTION NAME

Figure A.4

201	00000	337	FORALPSTIM_SER	(1,TOT_VAN,1,111N YEM_SER(TOT_VAN) IN CENRAL IN DISK, SUBSCRIPT 127
204	00000	340	FORACMSLAG	(1,TOT_VAN,1,111N LAG(TOT_VAN), SUBSCR,PT 75,76
176	00000	223	\$1C001	(*) INTERIM 104,110,174,176
177	00000	225	\$1C002	(*) INTERIM 110,175,177
179	00000	231	\$1C004	INTERIM 106,178,179
181	00000	236	\$1C005	(*) INTERIM 102,105,107,109,180,181
184	00000	266	\$1C010	INTERIM 97,102,104
186	00000	271	\$1C011	INTERIM 98,105,109
188	00000	276	\$1C012	INTERIM 95,107,108
190	00000	281	\$1C013	(*) INTERIM 95,107,150
192	00000	286	\$1C014	(*) INTERIM 93,151,152
194	00000	291	\$1C015	(*) INTERIM 92,153,154
196	00000	296	\$1C016	(*) INTERIM 91,155,198
8	340	63	ALPHA	(*) IN GROUP IN CLUP IN PARAM IN DISK, FIELD(DECIMAL(14,6)) 800,0,0,100,102,102,102
39	1050	93	AUTUM	IN DESCRIPTION IN REFLECT,DISK,DOCUMENT,REFLECT IN DOCUMENT IN DISK, FIELD(UNIQUE(10)) 137
13	450	9	DATA	IN DISK, FILE 14
142	00000	157	DATA	IN DISK IN EXIST, GROUP 103,107
10	340	62	DATA	(*) IN GROUP IN CLUP IN PARAM IN DISK, FIELD(DECIMAL(14,6)) 10,10,10,10,10,10,10,10
76	2270	75	C	(*) IN INTERIM(AMERIC(10)) 79,78,104,118
105	300	111	CAPITAL	ASSIGNMENT NAME
44	1350	15	CENTRAL_PARAM	IN INPUT IN TERM, RECORD 45,46,7,7,98

152	00000	147	C00E	IN TYPE IN TIM_SERIOT_VAN) IN CONTRAL IN DISK, FIELD(CHAR(4))
20	930	57	C00E	IN TYPE IN TIM_SERIOT-DISK.BANK-TIM_SER) IN BANK IN DISK, FIELD(CHAR(4))
2	320	20	C00P	IN PARAM IN DISK, GROUP 7,9,11
60	1000	42	C00P	IN REFLECTERIST-DISK.DOCUMENT-REFREC) IN DOCUMENT IN DISK, FIELD(CHAR(8))
7	330	22	C00P	IN CUFF IN PARAM IN DISK, RECORD 8
102	3000	110	C00P	ASSULTION NAME
171	00000	181	C00P	IN PCINTER, GROUP 172
105	00000	100	C00P	IN EXIST, GROUP 100
01	1100	7	C00P	IN DISK, FILE 42
100	00000	143	C00P	IN DISK, FILE 42
30	030	40	C00P	IN DISK, FILE 42
37	1030	45	C00P	IN DISK, FILE 42
34	1000	23	C00P	IN DISK, FILE 42
159	00000	124	C00P	IN DISK, FILE 42
160	00000	236	C00P	IN DISK, FILE 42
0	230	6	C00P	IN DISK, FILE 42
169	00000	177	C00P	IN DISK, FILE 42
160	00000	155	C00P	IN DISK, FILE 42
31	570	0	C00P	IN DISK, FILE 42
02	2370	72	C00P	IN DISK, FILE 42
107	00000	162	C00P	IN DISK, FILE 42

29	62C	50	ENDREF	IN TIM_SERIEIST.DISK.BANK.TIM_SER) IN BANK IN DISK, FIELD(CHAR(1))
56	157C	11	ENTRY	ITUTBIN OUTPUT(TIM_NP) IN REPEAT IN TERM, RECORD 60,1,102,63
158	000000	153	EXIST	GROUP 159,105
67	250C	67	G	(0,NL) INTERIMNUMERIC(10)) 87,71,96,114
12	30C	61	GAPMA	143IN "AGES" IN CUFF IN PARAM IN DISK, FIELD(DECIMAL(14,8)) 12,12,12,12,104,104,104,104
80	228C	74	I	(0,NL) INTERIMNUMERIC(10)) 60,90,103,105,119
50	241C	110	INCME	ASSERTION NAME
43	134C	6	INPLT	IN TERM, FILE 44,45
9	35C	21	INVESTM	IN CUFF IN PARAM IN DISK, RECORD 10
103	503C	113	INVESTMENT	ASSERTION NAME
104	000000	107	J	11,005_NUM,1,111IN DATA(US_NUM) IN TIM_SERIEIST.DISK.BANK.TIM_SER) IN PARAM IN DISK, SUBSCRIPT 95,6,69,124,130,131
85	24,0	69	A	(0,NL) INTERIMNUMERIC(10)) 85,105,9,103,105,105,124
154	000000	149	LABEL	14 NAME IN TIM_SERIEIST.VAR) IN CONTROL IN DISK, FIELD(CHAR(4))
57	155C	43	LABEL	IN TITLE IN OUTPUT(TIM_NP) IN REPEAT IN TERM, FIELD(PICTURE '00151A')
16	45C	60	LABEL	IN NAME IN TIM_SERIEIST.DISK.BANK.TIM_SER) IN BANK IN DISK, FIELD(CHAR(4)) 10,109
70	155C	76	LAB	(10UT_VAN) INTERIMNUMERIC(13)) 70,70,102,105,107
49	140J	14	LISTREP	IN INPUT IN TERM, RECORD 90,91
103	000000	202	ASALAG	INTERIM 75,76,77,78,95,96,97,103
1	13C	0	MECEL_I	MODULE NAME
106	3151	220	M	11,51A_P1,111IN OUTPUT(TIM_NP) IN REPORT IN TERM, SUBSCRIPT 107,108,109,110,114
142	000000	137	NAME	IN TIM_SERIEIST.VAR) IN CONTROL IN DISK, GROUP 134,135
15	400	27	NAME	IN TIM_SERIEIST.DISK.BANK.TIM_SER) IN BANK IN DISK, GROUP 10,17
140	000000	2	NU	INTERIA

A.4 (Cont.)

71	1551	221	NP	79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,117,118,119,120,121,122,123,124,125,126,127,128,129,130,131,132,133,134,135,136,137,138,139,140,141,142,143,144,145,146,147,148,149,150,151,152,153,154,155,156,157,158,159,160,161,162,163,164,165,166,167,168,169,170,171,172,173,174,175,176,177,178,179,180,181,182,183,184,185,186,187,188,189,190,191,192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,208,209,210,211,212,213,214,215,216,217,218,219,220,221,222,223,224,225,226,227,228,229,230,231,232,233,234,235,236,237,238,239,240,241,242,243,244,245,246,247,248,249,250,251,252,253,254,255,256,257,258,259,260,261,262,263,264,265,266,267,268,269,270,271,272,273,274,275,276,277,278,279,280,281,282,283,284,285,286,287,288,289,290,291,292,293,294,295,296,297,298,299,300,301,302,303,304,305,306,307,308,309,310,311,312,313,314,315,316,317,318,319,320,321,322,323,324,325,326,327,328,329,330,331,332,333,334,335,336,337,338,339,340,341,342,343,344,345,346,347,348,349,350,351,352,353,354,355,356,357,358,359,360,361,362,363,364,365,366,367,368,369,370,371,372,373,374,375,376,377,378,379,380,381,382,383,384,385,386,387,388,389,390,391,392,393,394,395,396,397,398,399,400,401,402,403,404,405,406,407,408,409,410,411,412,413,414,415,416,417,418,419,420,421,422,423,424,425,426,427,428,429,430,431,432,433,434,435,436,437,438,439,440,441,442,443,444,445,446,447,448,449,450,451,452,453,454,455,456,457,458,459,460,461,462,463,464,465,466,467,468,469,470,471,472,473,474,475,476,477,478,479,480,481,482,483,484,485,486,487,488,489,490,491,492,493,494,495,496,497,498,499,500,501,502,503,504,505,506,507,508,509,510,511,512,513,514,515,516,517,518,519,520,521,522,523,524,525,526,527,528,529,530,531,532,533,534,535,536,537,538,539,540,541,542,543,544,545,546,547,548,549,550,551,552,553,554,555,556,557,558,559,560,561,562,563,564,565,566,567,568,569,570,571,572,573,574,575,576,577,578,579,580,581,582,583,584,585,586,587,588,589,590,591,592,593,594,595,596,597,598,599,600,601,602,603,604,605,606,607,608,609,610,611,612,613,614,615,616,617,618,619,620,621,622,623,624,625,626,627,628,629,630,631,632,633,634,635,636,637,638,639,640,641,642,643,644,645,646,647,648,649,650,651,652,653,654,655,656,657,658,659,660,661,662,663,664,665,666,667,668,669,670,671,672,673,674,675,676,677,678,679,680,681,682,683,684,685,686,687,688,689,690,691,692,693,694,695,696,697,698,699,700,701,702,703,704,705,706,707,708,709,710,711,712,713,714,715,716,717,718,719,720,721,722,723,724,725,726,727,728,729,730,731,732,733,734,735,736,737,738,739,740,741,742,743,744,745,746,747,748,749,750,751,752,753,754,755,756,757,758,759,760,761,762,763,764,765,766,767,768,769,770,771,772,773,774,775,776,777,778,779,780,781,782,783,784,785,786,787,788,789,790,791,792,793,794,795,796,797,798,799,800,801,802,803,804,805,806,807,808,809,810,811,812,813,814,815,816,817,818,819,820,821,822,823,824,825,826,827,828,829,830,831,832,833,834,835,836,837,838,839,840,841,842,843,844,845,846,847,848,849,850,851,852,853,854,855,856,857,858,859,860,861,862,863,864,865,866,867,868,869,870,871,872,873,874,875,876,877,878,879,880,881,882,883,884,885,886,887,888,889,890,891,892,893,894,895,896,897,898,899,900,901,902,903,904,905,906,907,908,909,910,911,912,913,914,915,916,917,918,919,920,921,922,923,924,925,926,927,928,929,930,931,932,933,934,935,936,937,938,939,940,941,942,943,944,945,946,947,948,949,950,951,952,953,954,955,956,957,958,959,960,961,962,963,964,965,966,967,968,969,970,971,972,973,974,975,976,977,978,979,980,981,982,983,984,985,986,987,988,989,990,991,992,993,994,995,996,997,998,999,1000	IN ENTRY(TOT) IN OUTPUT(SIM_NP) IN REPORT IN TERM, SUBSCRIPT
151	00000	146	NUM_PD_YR	IN RANGE IN TIM_SER(TOT_VAM) IN CONTROL IN DISK, FIELD(12)	
27	000	52	NUM_PD_YR	IN RANGE IN TIM_SER(TOT_VAM) IN CONTROL IN DISK, FIELD(12)	
143	00000	138	NUMPER	IN TIM_SER(TOT_VAM) IN CONTROL IN DISK, FIELD(12)	
18	510	50	ALNUMBER	IN TIM_SER(TOT_VAM) IN CONTROL IN DISK, FIELD(12)	
61	1540	30	U	IN TIM_SER(TOT_VAM) IN CONTROL IN DISK, FIELD(12)	
62	1000	29	U	IN TIM_SER(TOT_VAM) IN CONTROL IN DISK, FIELD(12)	
150	00000	145	GUS_NUM	IN RANGE IN TIM_SER(TOT_VAM) IN CONTROL IN DISK, FIELD(12)	
26	560	53	GUS_NUM	IN RANGE IN TIM_SER(TOT_VAM) IN CONTROL IN DISK, FIELD(12)	
68	1570	78	OFFSET	IN RANGE IN TIM_SER(TOT_VAM) IN CONTROL IN DISK, FIELD(12)	
63	1810	28	OUT_VALUE	IN RANGE IN TIM_SER(TOT_VAM) IN CONTROL IN DISK, FIELD(12)	
60	1560	31	OUT_YEAR	IN RANGE IN TIM_SER(TOT_VAM) IN CONTROL IN DISK, FIELD(12)	
53	1510	12	OUTPUT	IN RANGE IN TIM_SER(TOT_VAM) IN CONTROL IN DISK, FIELD(12)	
83	2380	71	P	IN RANGE IN TIM_SER(TOT_VAM) IN CONTROL IN DISK, FIELD(12)	
5	310	10	PARAM	IN RANGE IN TIM_SER(TOT_VAM) IN CONTROL IN DISK, FIELD(12)	
157	00000	152	PERICO	IN RANGE IN TIM_SER(TOT_VAM) IN CONTROL IN DISK, FIELD(12)	
56	1540	34	PERIOD	IN RANGE IN TIM_SER(TOT_VAM) IN CONTROL IN DISK, FIELD(12)	
25	580	54	PERIOD	IN RANGE IN TIM_SER(TOT_VAM) IN CONTROL IN DISK, FIELD(12)	
168	00000	176	PCINTER	IN RANGE IN TIM_SER(TOT_VAM) IN CONTROL IN DISK, FIELD(12)	

A.4 (Cont.)

99	293C	117	PRIV_PROD	ASSERTION NAME	
104	3C6C	112	PRIV_WAGE	ASSERTION NAME	
101	257C	115	PMDEITS	ASSERTION NAME	
38	104C	44	PUBLISPER	IN DESCRIPTION IN REF(EXIST-DISK.DOCUMENT-REFEC) IN DOCUMENT IN DISK, FIELD(CHART20)	136
88	251C	66	R	(6,6C) INTERIM(NUMERIC(10))	88,92,99,99,115
145	000000	140	RANGE	IN TIM_SERITOT_VARI IN CONTROL IN DISK, GROUP	145,150,151
22	520	25	RANGE	IN TIM_SERITOT_VARI IN BANK IN DISK, GROUP	25,26,27
167	000000	172	REF#	(6,6C) BANK IN DISK IN EXIST, FIELD	26,146,149,167
166	000000	169	REF#	(6,6C) CONTROL IN EXIST, FIELD	129,130,140,166,168
166	000000	161	REF#	(6,6C) TIM_SERITOT_VARI IN CONTROL IN DISK, FIELD(NUMERIC(3))	127,131,146
33	950	48	REF#	IN REF(EXIST-DISK.DOCUMENT-REFEC) IN DOCUMENT IN DISK, FIELD(NUMERIC(3))	132
26	610	51	REF#	(6,6C) TIM_SERITOT_VARI IN BANK IN DISK, FIELD(NUMERIC(3))	32,127,179
170	000000	173	REFREC	(6,6C) IN DOCUMENT IN PCINTER, FIELD	32,127,179
161	000000	156	REFREC	IN DOCUMENT IN DISK IN EXIST, INTERIM	32
32	540	17	REFREC	EXIST-DISK.DOCUMENT-REFEC IN DOCUMENT IN DISK, RECORD	33,34,40
52	150C	5	REFRAT	IN TERM, FILE	53
65	155C	77	SAMPLE	(6,6C) VARI INTERIM(NUMERIC(3))	65,74,85,151,153,165
47	1360	39	SIM_MP	IN UNTAL_PARAM IN INPUT IN TERM, FIELD(NUMERIC(12))	47,53,55,71,71,77,113
46	137C	40	SIM_PD	IN UNTAL_PARAM IN INPUT IN TERM, FIELD(NUMERIC(12))	46,73,107,112,176
46	136C	36	SIM_TYPE	IN UNTAL_PARAM IN INPUT IN TERM, FIELD(CHART(12))	
45	136C	41	SIM_YEAR	IN UNTAL_PARAM IN INPUT IN TERM, FIELD(NUMERIC(4))	45,73,107,111
149	000000	144	START	IN RANGE IN TIM_SERITOT_VARI IN CONTROL IN DISK, GROUP	156,157

A.4 (Cont.)

23	56J	24	START	IN RANGE IN TIM_SER(EXIST-DISK-BANK-TIM_SER) IN BANK IN DISK, GROUP
				24,25
173	00000	107	T	(*) INTERIM 101,102,103,104,105,114,115,116,117,118,119,120,121,122,123,124,125,173 16,91,92,93,94,98,99,100,175,180
197	00000	333	TERM	MEDIA 52
191	00000	3	TERM	MEDIA 43
172	00000	102	TIM_SER	(*) IN CONTROL IN POINTER, FIELD 42,126,172
163	00000	156	TIP_SER	IN BANK IN DISK IN EXIST, INTERIM 15,72,140
42	115C	16	TIM_SER	(TOT_VARI IN CONTROL IN DISK, RECORD 102,143,144,145,146,147,148
14	47Q	18	TIP_SER	(EXIST-DISK-BANK-TIM_SER IN BANK IN DISK, RECORD 15,16,19,22,26,25,33
155	00000	150	TITLE	IN NAME IN TIM_SER(TOT_VARI) IN CONTROL IN DISK, FIELD(CHAR(40)) 135
35	101C	47	TITLE	IN DESCRIPTION IN REPAU(EXIST-DISK-DOCUMENT,ALFACU) IN DOCUMENT IN DISK, FIELD(CHAR(40)) 135
17	50Q	54	TITLE	IN NAME IN TIM_SER(EXIST-DISK-BANK-TIM_SER) IN BANK IN DISK, FIELD(CHAR(40)) 135
54	152C	13	TITLE	IN OUTPUT(SIM_NP) IN REPORT IN TERM, RECORD 59,20,57,28
139	00000	1	TOT_VAN	INTERIM 42,42,64,68,65,65,70,70,72,139
100	245C	116	TOT_NAGE	ASSENTILN NAME 51,59,106
50	141U	37	IG10	IN LISTREP IN INPUT IN TERM, FIELD(NUMERIC(4)) 51,59,106
144	00000	139	TYPE	IN TIM_SER(TOT_VARI) IN CONTROL IN DISK, GROUP 152,153
19	52Q	26	TYPE	IN TIM_SER(EXIST-DISK-BANK-TIM_SER) IN BANK IN DISK, GROUP 20,41
89	252C	65	T1	(*) INTERIM(NUMERIC(10)) 89,53,104,116
153	00000	140	UNITS	IN TYPE IN TIM_SER(TOT_VARI) IN CONTROL IN DISK, FIELD(CHAR(20))
21	54U	56	UNITS	IN TYPE IN TIM_SER(EXIST-DISK-BANK-TIM_SER) IN BANK IN DISK, FIELD(CHAR(20))
54	156C	32	VALUE	IN TITLE IN OUTPUT(SIM_NP) IN REPORT IN TERM, FIELD(PICTURE '99(5)A') 58,67
51	142Q	36	VAR#	(TOT) IN LISTREP IN INPUT IN TERM, FIELD(NUMERIC(4)) 51,174

A.4 (Cont.)

91	261C	125	VAR#1	ASSERTION NAME
92	264C	124	VAR#2	ASSERTION NAME
93	2630	123	VAR#3	ASSERTION NAME
94	264C	122	VAR#4	ASSERTION NAME
95	1J2C	46	VCL#	IN DESCRIPTION IN AEPHICERIST-DISK.DOCUMENT.NEPAEC IN DOCUMENT IN DISK. FIELD(GH#4(4)) 134
96	255C	70	M	(P,NU) INTERMINUMERIC(1101) 84,100,101,102,123
97	37C	14	MA#5	IN COEFF IN PARAM IN OL-K, RECORD 12
98	224C	73	M1	(P,NU) INTERMINUMERIC(1101) 01,100,104,120
99	253C	64	M2	(P,NU) INTERMINUMERIC(1101) 90,94,97,100,117
100	2420	68	V	(P,NU) INTERMINUMERIC(1101) 86,90,94,101,125
101	000000	131	YEAR	IN START IN RANGE IN TIM_SKEUT_VAN IN CONTROL IN DISK, FIELD(NUPEIC(4)) 111,130
102	153C	35	YEAR	IN TITLE IN OUTPUT(SM_NP) IN REPORT IN TERM, FIELD(PICTURE *1434*) 55,64
103	57C	55	YEAR	IN START IN RANGE IN TIM_SERIEAIST-DISK.BANK.TIM_SER# IN BANK IN DISK. FIELD(NUPEIC(5)) 24,73

A.4 (Cont.)

PRECEDENCE MATRIX									
DICTO SUCCESSORS					PRECESSIONS				
1	70(0)	77(0)	70(0)	10(0)	300(2)				
2	60(0)	65(0)	64(0)	67(0)	71(0)	300(2)			
3	72(0)	73(0)	75(0)	73(0)					
4	81(0)	91(0)	10(1)						
5	32(0)								
6	14(0)	15(0)							
7	33(0)								
8	17(0)								
9	10(0)	8(0)							
10	20(0)	5(0)							
11	14(0)								
12	5(0)								
13	12(0)	11(0)							
14	30(0)	37(0)							
15	30(0)	30(0)	60(0)	10(0)					
16	7(0)								
17	23(0)	62(0)	40(0)						
18	25(0)	26(0)	27(0)	50(0)	50(0)				
19	61(0)								
20	10(0)	21(0)	24(0)						
21	62(0)	19(0)							
22	63(0)	23(0)							
23	43(0)	64(0)	45(0)	67(0)	62(0)				
24	50(0)	55(0)	53(0)						
25	20(0)	52(0)	51(0)						
26	50(0)	37(0)	25(0)						
27	50(0)	60(0)	50(0)						
28	11(0)	26(0)							
29	11(0)	25(0)							
30	11(0)	32(0)							
31	11(0)	32(0)							
32	13(0)	32(0)							
33	13(0)	32(0)							
34	13(0)	32(0)							
35	13(0)	32(0)							
36	13(0)	32(0)							
37	22(0)	11(0)	24(0)						
38	22(0)	12(0)	21(0)	30(0)					
39	21(0)	25(0)							
40	21(0)	25(0)							
41	21(0)	25(0)							
42	21(0)	25(0)							
43	21(0)	25(0)							
44	21(0)	25(0)							
45	21(0)	25(0)							
46	21(0)	25(0)							
47	21(0)	25(0)							
48	21(0)	25(0)							
49	21(0)	25(0)	20(0)	20(0)	290(0)	290(0)			
50	65(0)								

Figure A.5

51	50(9)	300(11)	52(9)	18(11)	172(10)	25(9)
52	23(11)	45(10)		25(11)	53(9)	
53	42(10)	45(10)		25(11)	24(9)	
54	30(11)	54(9)		24(11)	55(9)	
55	30(11)	54(9)		20(11)	57(9)	
56	36(9)			20(11)		
57	36(9)	26(9)		20(11)	27(9)	
58	104(11)			18(11)	60(9)	
59				27(11)		
60	220(11)	58(9)		27(11)		
61	44(11)	24(11)	242(11)	44(11)		
62	247(11)	242(11)	251(11)	24(11)		
63	253(11)	254(11)	255(11)	24(11)		
64				24(11)		
65				24(11)		
66				24(11)		
67				24(11)		
68				24(11)		
69				24(11)		
70				24(11)		
71				24(11)		
72				24(11)		
73				24(11)		
74				24(11)		
75				24(11)		
76				24(11)		
77				24(11)		
78				24(11)		
79				24(11)		
80				24(11)		
81				24(11)		
82				24(11)		
83				24(11)		
84				24(11)		
85				24(11)		
86				24(11)		
87				24(11)		
88				24(11)		
89				24(11)		
90				24(11)		
91				24(11)		
92				24(11)		
93				24(11)		
94				24(11)		
95				24(11)		
96				24(11)		
97				24(11)		
98				24(11)		
99				24(11)		
100				24(11)		
101				24(11)		
102				24(11)		
103				24(11)		
104				24(11)		
105				24(11)		
106				24(11)		
107				24(11)		
108				24(11)		
109				24(11)		
110				24(11)		
111				24(11)		
112				24(11)		
113				24(11)		
114				24(11)		
115				24(11)		
116				24(11)		
117				24(11)		
118				24(11)		
119				24(11)		
120				24(11)		
121				24(11)		
122				24(11)		
123				24(11)		
124				24(11)		
125				24(11)		
126				24(11)		
127				24(11)		
128				24(11)		
129				24(11)		
130				24(11)		
131				24(11)		
132				24(11)		
133				24(11)		
134				24(11)		
135				24(11)		
136				24(11)		
137				24(11)		
138				24(11)		
139				24(11)		
140				24(11)		
141				24(11)		
142				24(11)		
143				24(11)		
144				24(11)		
145				24(11)		
146				24(11)		
147				24(11)		
148				24(11)		
149				24(11)		
150				24(11)		
151				24(11)		
152				24(11)		
153				24(11)		
154				24(11)		
155				24(11)		
156				24(11)		
157				24(11)		
158				24(11)		
159				24(11)		
160				24(11)		
161				24(11)		
162				24(11)		
163				24(11)		
164				24(11)		
165				24(11)		
166				24(11)		
167				24(11)		
168				24(11)		
169				24(11)		
170				24(11)		
171				24(11)		
172				24(11)		
173				24(11)		
174				24(11)		
175				24(11)		
176				24(11)		
177				24(11)		
178				24(11)		
179				24(11)		
180				24(11)		
181				24(11)		
182				24(11)		
183				24(11)		
184				24(11)		
185				24(11)		
186				24(11)		
187				24(11)		
188				24(11)		
189				24(11)		
190				24(11)		
191				24(11)		
192				24(11)		
193				24(11)		
194				24(11)		
195				24(11)		
196				24(11)		
197				24(11)		
198				24(11)		
199				24(11)		
200				24(11)		
201				24(11)		
202				24(11)		
203				24(11)		
204				24(11)		
205				24(11)		
206				24(11)		
207				24(11)		
208				24(11)		
209				24(11)		
210				24(11)		
211				24(11)		
212				24(11)		
213				24(11)		
214				24(11)		
215				24(11)		
216				24(11)		
217				24(11)		
218				24(11)		
219				24(11)		
220				24(11)		
221				24(11)		
222				24(11)		
223				24(11)		
224				24(11)		
225				24(11)		
226				24(11)		
227				24(11)		
228				24(11)		
229				24(11)		
230				24(11)		
231				24(11)		
232				24(11)		
233				24(11)		
234				24(11)		
235				24(11)		
236				24(11)		
237				24(11)		
238				24(11)		
239				24(11)		
240				24(11)		
241				24(11)		
242				24(11)		
243				24(11)		
244				24(11)		
245				24(11)		
246				24(11)		
247				24(11)		
248				24(11)		
249				24(11)		
250				24(11)		
251				24(11)		
252				24(11)		
253				24(11)		
254				24(11)		
255				24(11)		
256				24(11)		
257				24(11)		
258				24(11)		
259				24(11)		
260				24(11)		
261				24(11)		
262				24(11)		
263				24(11)		
264				24(11)		
265				24(11)		
266				24(11)		
267				24(11)		
268				24(11)		
269				24(11)		
270				24(11)		
271				24(11)		
272				24(11)		
273				24(11)		
274				24(11)		
275				24(11)		
276				24(11)		
277				24(11)		
278				24(11)		
279				24(11)		
280				24(11)		
281				24(11)		
282				24(11)		
283				24(11)		
284				24(11)		
285				24(11)		
286				24(11)		
287				24(11)		
288				24(11)		
289				24(11)		
290				24(11)		
291				24(11)		
292				24(11)		
293				24(11)		
294				24(11)		
295				24(11)		
296				24(11)		
297				24(11)		
298				24(11)		
299				24(11)		
300				24(11)		
301				24(11)		
302				24(11)		
303				24(11)		
304				24(11)		
305				24(11)		
306				24(11)		
307				24(11)		
308				24(11)		
309				24(11)		
310				24(11)		
311				24(11)		
312				24(11)		
313				24(11)		
314				24(11)		
315				24(11)		
316				24(11)		
317				24(11)		
318				24(11)		
319				24(11)		
320				24(11)		
321				24(11)		
322				24(11)		
323				24(11)		
324				24(11)		
325				24(11)		
326				24(11)		
327				2		

111	236(4)	188(3)	202(3)	237(3)	239(3)	244(3)	239(3)	245(3)
112	240(4)	189(3)	241(3)	242(3)	196(3)	243(3)	244(3)	245(3)
113	246(4)	190(3)	247(3)	248(3)	194(3)	249(3)	250(3)	251(3)
114	252(4)	191(3)	253(3)	254(3)	192(3)	255(3)	194(3)	256(3)
115	257(4)	192(3)	196(3)	192(3)				
116	258(4)	193(3)	197(3)	204(3)				
117	259(4)	194(3)	198(3)	205(3)	404(3)			
118	260(4)	195(3)	199(3)	206(3)	208(3)			
119	261(4)	196(3)	200(3)	207(3)	209(3)			
120	262(4)	197(3)	201(3)	208(3)	210(3)			
121	263(4)	198(3)	202(3)	209(3)	211(3)			
122	264(4)	199(3)	203(3)	210(3)	212(3)			
123	265(4)	200(3)	204(3)	211(3)	213(3)			
124	266(4)	201(3)	205(3)	212(3)	214(3)			
125	267(4)	202(3)	206(3)	213(3)	215(3)			
126	268(4)	203(3)	207(3)	214(3)	216(3)			
127	269(4)	204(3)	208(3)	215(3)	217(3)			
128	270(4)	205(3)	209(3)	216(3)	218(3)			
129	271(4)	206(3)	210(3)	217(3)	219(3)			
130	272(4)	207(3)	211(3)	218(3)	220(3)			
131	273(4)	208(3)	212(3)	219(3)	221(3)			
132	274(4)	209(3)	213(3)	220(3)	222(3)			
133	275(4)	210(3)	214(3)	221(3)	223(3)			
134	276(4)	211(3)	215(3)	222(3)	224(3)			
135	277(4)	212(3)	216(3)	223(3)	225(3)			
136	278(4)	213(3)	217(3)	224(3)	226(3)			
137	279(4)	214(3)	218(3)	225(3)	227(3)			
138	280(4)	215(3)	219(3)	226(3)	228(3)			
139	281(4)	216(3)	220(3)	227(3)	229(3)			
140	282(4)	217(3)	221(3)	228(3)	230(3)			
141	283(4)	218(3)	222(3)	229(3)	231(3)			
142	284(4)	219(3)	223(3)	230(3)	232(3)			
143	285(4)	220(3)	224(3)	231(3)	233(3)			
144	286(4)	221(3)	225(3)	232(3)	234(3)			
145	287(4)	222(3)	226(3)	233(3)	235(3)			
146	288(4)	223(3)	227(3)	234(3)	236(3)			
147	289(4)	224(3)	228(3)	235(3)	237(3)			
148	290(4)	225(3)	229(3)	236(3)	238(3)			
149	291(4)	226(3)	230(3)	237(3)	239(3)			
150	292(4)	227(3)	231(3)	238(3)	240(3)			
151	293(4)	228(3)	232(3)	239(3)	241(3)			
152	294(4)	229(3)	233(3)	240(3)	242(3)			
153	295(4)	230(3)	234(3)	241(3)	243(3)			
154	296(4)	231(3)	235(3)	242(3)	244(3)			
155	297(4)	232(3)	236(3)	243(3)	245(3)			
156	298(4)	233(3)	237(3)	244(3)	246(3)			
157	299(4)	234(3)	238(3)	245(3)	247(3)			
158	300(4)	235(3)	239(3)	246(3)	248(3)			
159	301(4)	236(3)	240(3)	247(3)	249(3)			
160	302(4)	237(3)	241(3)	248(3)	250(3)			
161	303(4)	238(3)	242(3)	249(3)	251(3)			
162	304(4)	239(3)	243(3)	250(3)	252(3)			
163	305(4)	240(3)	244(3)	251(3)	253(3)			
164	306(4)	241(3)	245(3)	252(3)	254(3)			
165	307(4)	242(3)	246(3)	253(3)	255(3)			
166	308(4)	243(3)	247(3)	254(3)	256(3)			

167	86(3)	87(3)	88(3)	119(3)	120(3)	121(3)			
168	153(2)	154(3)					169(2)	171(2)	
169	166(2)	141(6)					87(4)		
170	165(2)						88(4)		
171	164(2)						174(2)		
172	151(2)	51(6)	158(5)				174(8)		
173	66(3)						69(4)		
174	172(2)	173(6)							
175	84(3)	152(3)							
176							177(2)	181(2)	
177	176(2)						174(2)	181(9)	
178	171(2)	171(7)					174(2)		
179	176(2)						150(4)		
180	54(3)						150(4)		
181	171(2)	177(9)					150(2)		
182	154(2)	161(7)					151(2)		
183	152(2)						91(4)		
184	114(3)						201(1)		
185	14-1(2)						92(4)		
186	14-1(2)						260(6)		
187	94(3)	115(3)	117(3)				301(2)		
188	94(3)	53(3)	94(3)	95(3)	54(3)	97(3)	58(3)	99(3)	
189	130(3)	131(3)	162(3)	102(3)	111(3)	112(3)	113(3)	114(3)	
190	115(3)	116(3)	117(3)	118(3)	122(3)	123(3)	124(3)	125(3)	
191	235(3)								
192	183(2)								
193	183(2)	112(3)					93(4)	264(8)	
194	183(3)	113(3)	114(3)				236(8)	93(4)	
195	183(2)						256(8)		
196	56(3)	112(3)					93(4)		
197	183(2)						257(6)	269(8)	
198	57(3)	116(3)					56(4)		
199	183(2)						257(8)	274(8)	
200	183(3)	111(3)	116(3)				57(4)		
201	183(3)						257(6)		
202	183(3)	118(3)					257(8)		
203	183(3)						100(8)		
204	183(3)	116(3)	117(3)				279(6)		
205	183(2)						101(4)		
206	183(2)	112(3)					284(6)		
207	183(2)						102(4)		
208	183(3)	117(3)	118(3)				284(8)		
209	183(2)	226(9)					103(4)		
210	183(3)	118(3)					284(6)		
211	183(6)	55(6)	145(2)				104(4)		
212	183(3)	127(3)	221(6)	12(6)			39(11)		
213	183(2)						105(4)		
214	183(3)	115(3)	131(3)	285(3)			42(11)		
215	183(2)						106(4)		
216	183(3)	115(3)	131(3)				41(11)		
217	183(2)						332(3)	250(3)	
218	183(2)						332(3)	221(3)	
219	26(2)						107(4)		
220	183(3)	108(3)	105(3)	115(3)	217(3)		37(8)		
221	183(3)	108(3)	105(3)	115(3)	128(3)	130(3)	38(8)	212(6)	
222	183(3)						209(8)		
223							331(2)		

A.5 (Cont.)

224	107(3)	108(3)	351(8)
225			329(2)
226	107(3)		329(8)
227	25(2)		305(4)
228	138(3)		63(1)
229	220(4)		214(3)
230	35(2)		109(4)
231			328(2)
232	105(3)		528(8)
233	105(3)		52(1)
234	31(2)		113(4)
235	327(4)		183(3)
236	65(2)		111(4)
237	111(3)		204(8)
238	111(4)	112(3)	327(2)
239	111(4)	113(3)	327(8)
240	71(2)	139(2)	127(1)
241	112(3)		61(1)
242	112(3)		61(1)
243	112(3)		274(8)
244	112(3)		61(1)
245	112(3)		61(1)
246	75(2)	206(8)	113(4)
247	113(3)		62(1)
248	113(3)		62(1)
249	113(3)		62(1)
250	113(3)	114(3)	209(8)
251	113(3)		62(1)
252	75(2)	202(8)	115(4)
253	114(3)		63(1)
254	114(3)		63(1)
255	114(3)		63(1)
256	114(3)		115(4)
257	71(2)	155(8)	120(2)
258	71(2)	162(8)	117(4)
259	71(2)	164(8)	117(4)
260	65(2)	100(8)	328(1)
261	225(4)		328(2)
262	115(3)	120(3)	332(8)
263	115(3)	121(3)	332(8)
264	65(2)	154(8)	117(4)
265	115(3)	237(8)	329(1)
266			329(2)
267	115(3)		329(8)
268	223(4)		324(3)
269	71(2)	194(3)	123(4)
270	125(3)		49(1)
271			323(2)
272	125(3)		323(8)
273	221(4)		322(3)
274	72(2)	156(8)	121(4)
275	121(3)		49(1)
276			321(2)
277	121(3)		321(8)
278	319(4)		320(3)
279	64(2)	204(8)	122(4)
280	122(3)		49(1)
281			319(4)
282	122(3)		319(8)
283	317(4)		318(3)

284	451(2)	206(10)	123(4)
285	123(3)		49(1)
286			317(1)
287	123(3)		317(2)
288	215(6)		317(3)
289	66(2)	208(10)	126(2)
290	126(3)		49(1)
291			315(1)
292	126(3)		315(2)
293	213(6)		315(3)
294	67(2)	412(10)	318(3)
295	125(3)		125(4)
296			48(1)
297	125(3)		312(2)
298	70(2)		312(3)
299	126(3)	130(13)	126(4)
300	65(1)	55(6)	68(6)
301	126(3)	71(6)	74(6)
302	126(3)	100(10)	332(10)
303	77(2)	314(4)	
304	77(2)	314(4)	
305	77(2)	295(10)	
306	121(3)		
307	121(3)		
308	76(1)	77(6)	78(6)
309	34(2)		106(6)
310	34(2)		112(2)
311	34(2)		
312	25(2)		
313	25(2)	257(10)	
314	25(2)		
315	25(2)	252(10)	
316	25(2)		
317	25(2)	267(10)	
318	25(2)		
319	25(2)	262(10)	
320	25(2)		
321	25(2)	277(10)	
322	25(2)		
323	25(2)	272(10)	
324	25(2)		
325	25(2)	267(10)	
326	25(2)		
327	25(2)	235(10)	
328	25(2)	432(10)	
329	25(2)	225(10)	
330	25(2)		
331	25(2)	228(10)	
332	25(2)		
333	25(2)		
334	25(2)		
335	25(2)		
336	25(2)		
337	25(2)		
338	25(2)		
339	25(2)	50(1)	54(1)
340	126(3)	106(1)	106(1)
341	126(3)	106(1)	106(1)
342	126(3)	106(1)	106(1)
343	126(3)	106(1)	106(1)
344	126(3)	106(1)	106(1)
345	126(3)	106(1)	106(1)
346	126(3)	106(1)	106(1)
347	126(3)	106(1)	106(1)
348	126(3)	106(1)	106(1)
349	126(3)	106(1)	106(1)
350	126(3)	106(1)	106(1)
351	126(3)	106(1)	106(1)
352	126(3)	106(1)	106(1)
353	126(3)	106(1)	106(1)
354	126(3)	106(1)	106(1)
355	126(3)	106(1)	106(1)
356	126(3)	106(1)	106(1)
357	126(3)	106(1)	106(1)
358	126(3)	106(1)	106(1)
359	126(3)	106(1)	106(1)
360	126(3)	106(1)	106(1)
361	126(3)	106(1)	106(1)
362	126(3)	106(1)	106(1)
363	126(3)	106(1)	106(1)
364	126(3)	106(1)	106(1)
365	126(3)	106(1)	106(1)
366	126(3)	106(1)	106(1)
367	126(3)	106(1)	106(1)
368	126(3)	106(1)	106(1)
369	126(3)	106(1)	106(1)
370	126(3)	106(1)	106(1)
371	126(3)	106(1)	106(1)
372	126(3)	106(1)	106(1)
373	126(3)	106(1)	106(1)
374	126(3)	106(1)	106(1)
375	126(3)	106(1)	106(1)
376	126(3)	106(1)	106(1)
377	126(3)	106(1)	106(1)
378	126(3)	106(1)	106(1)
379	126(3)	106(1)	106(1)
380	126(3)	106(1)	106(1)
381	126(3)	106(1)	106(1)
382	126(3)	106(1)	106(1)
383	126(3)	106(1)	106(1)
384	126(3)	106(1)	106(1)
385	126(3)	106(1)	106(1)
386	126(3)	106(1)	106(1)
387	126(3)	106(1)	106(1)
388	126(3)	106(1)	106(1)
389	126(3)	106(1)	106(1)
390	126(3)	106(1)	106(1)
391	126(3)	106(1)	106(1)
392	126(3)	106(1)	106(1)
393	126(3)	106(1)	106(1)
394	126(3)	106(1)	106(1)
395	126(3)	106(1)	106(1)
396	126(3)	106(1)	106(1)
397	126(3)	106(1)	106(1)
398	126(3)	106(1)	106(1)
399	126(3)	106(1)	106(1)
400	126(3)	106(1)	106(1)
401	126(3)	106(1)	106(1)
402	126(3)	106(1)	106(1)
403	126(3)	106(1)	106(1)
404	126(3)	106(1)	106(1)
405	126(3)	106(1)	106(1)
406	126(3)	106(1)	106(1)
407	126(3)	106(1)	106(1)
408	126(3)	106(1)	106(1)
409	126(3)	106(1)	106(1)
410	126(3)	106(1)	106(1)
411	126(3)	106(1)	106(1)
412	126(3)	106(1)	106(1)
413	126(3)	106(1)	106(1)
414	126(3)	106(1)	106(1)
415	126(3)	106(1)	106(1)
416	126(3)	106(1)	106(1)
417	126(3)	106(1)	106(1)
418	126(3)	106(1)	106(1)
419	126(3)	106(1)	106(1)
420	126(3)	106(1)	106(1)
421	126(3)	106(1)	106(1)
422	126(3)	106(1)	106(1)
423	126(3)	106(1)	106(1)
424	126(3)	106(1)	106(1)
425	126(3)	106(1)	106(1)
426	126(3)	106(1)	106(1)
427	126(3)	106(1)	106(1)
428	126(3)	106(1)	106(1)
429	126(3)	106(1)	106(1)
430	126(3)	106(1)	106(1)
431	126(3)	106(1)	106(1)
432	126(3)	106(1)	106(1)
433	126(3)	106(1)	106(1)
434	126(3)	106(1)	106(1)
435	126(3)	106(1)	106(1)
436	126(3)	106(1)	106(1)
437	126(3)	106(1)	106(1)
438	126(3)	106(1)	106(1)
439	126(3)	106(1)	106(1)
440	126(3)	106(1)	106(1)
441	126(3)	106(1)	106(1)
442	126(3)	106(1)	106(1)
443	126(3)	106(1)	106(1)
444	126(3)	106(1)	106(1)
445	126(3)	106(1)	106(1)
446	126(3)	106(1)	106(1)
447	126(3)	106(1)	106(1)
448	126(3)	106(1)	106(1)
449	126(3)	106(1)	106(1)
450	126(3)	106(1)	106(1)
451	126(3)	106(1)	106(1)
452	126(3)	106(1)	106(1)
453	126(3)	106(1)	106(1)
454	126(3)	106(1)	106(1)
455	126(3)	106(1)	106(1)
456	126(3)	106(1)	106(1)
457	126(3)	106(1)	106(1)
458	126(3)	106(1)	106(1)
459	126(3)	106(1)	106(1)
460	126(3)	106(1)	106(1)
461	126(3)	106(1)	106(1)
462	126(3)	106(1)	106(1)
463	126(3)	106(1)	106(1)
464	126(3)	106(1)	106(1)
465	126(3)	106(1)	106(1)
466	126(3)	106(1)	106(1)
467	126(3)	106(1)	106(1)
468	126(3)	106(1)	106(1)
469	126(3)	106(1)	106(1)
470	126(3)	106(1)	106(1)
471	126(3)	106(1)	106(1)
472	126(3)	106(1)	106(1)
473	126(3)	106(1)	106(1)
474	126(3)	106(1)	106(1)
475	126(3)	106(1)	106(1)
476	126(3)	106(1)	106(1)
477	126(3)	106(1)	106(1)
478	126(3)	106(1)	106(1)
479	126(3)	106(1)	106(1)
480	126(3)	106(1)	106(1)
481	126(3)	106(1)	106(1)
482	126(3)	106(1)	106(1)
483	126(3)	106(1)	106(1)
484	126(3)	106(1)	106(1)
485	126(3)	106(1)	106(1)
486	126(3)	106(1)	106(1)
487	126(3)	106(1)	106(1)
488	126(3)	106(1)	106(1)
489	126(3)	106(1)	106(1)
490	126(3)	106(1)	106(1)
491	126(3)	106(1)	106(1)
492	126(3)	106(1)	106(1)
493	126(3)	106(1)	106(1)
494	126(3)	106(1)	106(1)
495	126(3)	106(1)	106(1)
496	126(3)	106(1)	106(1)
497	126(3)	106(1)	106(1)
498	126(3)	106(1)	106(1)
499	126(3)	106(1)	106(1)
500	126(3)	106(1)	106(1)

A.5 (Cont.)

342 133(3) 134(3) 135(3) 136(3)

A.5 (Cont.)

APPENDIX B

This Appendix present the complete PL/1 source listing of the Cycles Processor. The main Cycles procedure is used as a monitor which calls on each of the internally defined other components of the processor. The following procedures are used:

<u>NAME</u>	<u>FUNCTION</u>
CYCLES	Main monitor.
STRGCON	Depth-First Linear Algorithm to find strongly connected components (S-C-C).
PRCOMP	Printing routine for elements which are S-C-C.
ANALCMP	Procedure for analysis of S-C-C.
LINKAS	To "link" storage entries of S-C-C in the Associative Memory.
CHKMULT	Checks assertions with more than one target variable.
PSIMASS	Print simultaneous assertions under a common header in the specification list.
PRECED	Used to rank nodes of the Digraph.
RANK	Called by PRECED.

GENA 7963 05/05/78 PLIC-VER-033078
VERSION 5.5

05/30 PL/I COMPILER (F) ON VMS

PAGE
DATE 78.1

PL/I F COMPILER OPTIONS SPECIFIED ARE AS FOLLOWS--
A,X,MACHO

COMPILE-TIME MACRO PROCESSOR

COMPILER DIAGNOSTICS.

WARNINGS.

IFM30901 COMPILER CORE REQUIREMENT EXCEEDED SIZE GIVEN. AUXILIARY STORAGE USED.

END OF DIAGNOSTICS.

SOURCE LISTING.

SYMT LEVEL NEST

```

1      CYCLES: PROCEDURE (N,PCOMP,SUB);
      /*MAIN MONITOR FOR THE "CYCLES ANALYSIS" PROCESSOR.
      INITIALIZES VARIABLES,ALLOCATES SPACE AND CALLS
      PROCEDURE "STRUCON";
      ALGORITHM TO FIND STRONGLY CONNECTED COMPONENTS
      OF A GRAPH.*/
      /*REF: "DEPTH FIRST SEARCH AND LINEAR GRAPH ALGORITHMS"
      BY ROBERT TARJAN
      PUBLISHED IN SIAM VOL1, NO.2, JUNE 1972.*/
      /*THE GRAPH REPRESENTING RELATIONSHIPS BETWEEN DATA
      NAMES IS GIVEN BY AN ADJACENCY LIST STRUCTURE "ADJ".
      THE STRONGLY CONNECTED COMPONENTS REPRESENT EITHER
      CIRCULAR DEFINITIONS OR SETS OF SIMULTANEOUS ASSERTIONS
      TO BE GROUPED FOR SOLUTION BY THE SYSTEM.*/
      /*"N" IS THE NUMBER OF VERTICES OF THE GRAPH WHICH
      CORRESPONDS TO THE NUMBER OF DICTIONARY ENTRIES IN
      THE CONTEXT OF M O D E L SYSTEM.*/
      /*"PCOMP" IS A POINTER TO A LIST CONTAINING POINTERS
      TO EACH GROUP OF STRONGLY CONNECTED COMPONENTS
      "PTRLIST". IF NO CYCLES ARE FOUND PCOMP IS NULL */
      /*"SUB" IS A BIT PARAMETER. WHEN SET NO ANALYSIS IS
      DONE ON S-C-C AND THE LIST OF S-C-C IS NOT RELEASED.
      USED WHEN "CYCLES" IS CALLED FOR SUBSCRIPT ANALYSIS*/
2      1      DCL TRACEN ENTRY(0IN FIXED,CHAR(*),POINTER);
3      1      DCL TRACEX ENTRY(0IN FIXED,CHAR(*),POINTER,0IN FIXED);
4      1      DCL 1 ADJP_ENTRY BASED(ADJMP);

```

1
2
2
2
2
2
6
7
7
7
9
10
10
10
10
10
14
15
15
15
17
18
18
18
20
21
21
21
23
593
594
595

GANA 7463 US/05/78 PLIC-VLR-033078

PAGE

STMT LEVEL NEST

		2 ALLOC BIN FIXED,	596
		2 RUSED LIN FIXED, /*USED ROWS*/	597
		2 CUSED LIN FIXED, /*USED COLUMNS*/	598
		2 ENTRESCH REFER (ALLOC),	599
		3 COL LIN FIXED, /*COLUMN #*/	600
		/*SUCCESSOR COLUMN #*/	601
		3 EVALUF BIN FIXED, /*COLUMN VALUF*/	602
		3 ROW LIN FIXED, /*ROW #*/	603
		/*PREDCESSOR ROW #*/	604
		3 NVALUE BIN FIXED, /*ROW VALUE*/	605
5	1	DCL ACORNA ENTRY (BIN FIXED, POINTER);	26
6	1	DCL PRASSA ENTRY;	27
7	1	DCL ACKDR ENTRY (BIN FIXED, POINTER);	28
8	1	DCL FLAGLNK BIT(1) EXT INIT('0'B);	29
9	1	DCL FLAGLNK BIT(1) EXT INIT('0'B);	30
10	1	DCL FLAGLNK BIT(1) EXT INIT('0'B);	31
11	1	DCL PCOMP PTR;	32
12	1	DCL SUB BIT(1);	33
13	1	DCL ORDER(N) BIN FIXED EXT CTL;	34
		/*AMAP: ARRAY USED TO MAP ADJACENCY LIST INTO	35
		ACYCLIC BIT ADJACENCY MATRIX FOR ORDERING */	35
			36
14	1	DCL AMAP(N) BIN FIXED;	37
15	1	DCL 1 PTRLIST BASED (PCOMP),	38
		2 GROUPS C/N FIXED, /*NUMBER OF LISTS OF S-C-C*/	39
		2 PTRCMP (N, REFER (PGROUPS)) PTR;	40
		/*POINTERS TO EACH S-C-C LIST */	41
16	1	DCL 1 COMP BASED (P),	42
		2 INDEX BIN FIXED,	43
		2 LINK PTR, /*POINTER TO NEXT COMPONENT */	44
		2 LIST PTR, /*POINTER TO ADJACENCY LIST */	45
		2 KEY CHAR(2),	46
		2 STAT LITS BIT(7);	47
17	1	DCL 1 CMPLIST BASED (L),	48
		2 ADJNODE BIN FIXED,	49
		2 TYPE BIT FIXED,	50
		2 NEXT PTR;	51
18	1	DCL (P, PTRST, LAST, L) PTR;	52
19	1	DCL (LONLINK(N), NUMB(N), STACK(N)) BIN FIXED CTL ;	53
20	1	DCL (V, L, PTR, COUNT) BIN FIXED ;	54
21	1	DCL ONSTACK(N) BIT(1) CTL ;	55
22	1	DCL ONCMP(N) BIT(1) CTL ;	56
23	1	DCL FIRST_TIME PTR(1) INIT('1'P);	57
24	1	PUT SP IF LIST ('START CYCLES ANALYSIS');	58
25	1	ALLOCATE LONLINK, NUMB, STACK, ONSTACK, ONCMP ;	59
26	1	ALLOCATE ORDER;	60
27	1	COUNT=0;	61
28	1	AMAP=0;	62
29	1	PCOMP=NULL;	63
30	1	STACK, LONLINK, NUMB=0;	64
31	1	L, PTR=0;	65

GANA 7963 05/05/78 PLIC-VER-033078

PAGE

STPT LEVEL NEST

```

32 1          UNSTACK="0"B; 66
33 1          UNCMP="0"B; 67
34 1          DO V=1 TO N; 68
35 1 1        DO WHILE (NUMP(V)=0); 69
36 1 2        CALL STRGCCN (V,N); 70
37 1 2        END; 71
38 1 1        END; 72
39 1          FREE L->LINK,NUMB,STACK,ONSTACK,ONCMP; 73
40 1          IF SUB THEN RETURN; 74
/* 75
* 75
* PRINT ASSERTIONS 75
* 75
*/ 75
42 1          CALL PRTOSSX; 79
43 1          IF PCOMP = NULL THEN CALL PSIMASS; 80
44 1          CALL PRECED(N); 81
45 1          CALL PRECED(N); 82
46 1          *LSECOMP: ENTRY (N,PCOMP); 83
47 1          IF (PCOMP=NULL & FLAGRLS) THEN DO; 84
48 1 1        DO WHILE (#GROUPS>0); 85
49 1 2        FIRST = PTRCMP(#GROUPS); 86
50 1 2        DO WHILE (FIRST=NULL); 87
51 1 3        P=FIRST; 88
52 1 3        IF "SUP THEN DO WHILE (P->LIST=NULL); 89
53 1 4        L=P->LIST; 90
54 1 4        P->LIST=L->NEXT; 91
55 1 4        FREE L->CMPLIST; 92
56 1 4        END; 93
57 1 3        FIRST=P->LINK; 94
58 1 3        FREE P->COMP; 95
59 1 3        END; 96
60 1 2        #GROUPS=#GROUPS-1; 97
61 1 2        END; 98
62 1 1        #GROUPS=N; 99
63 1 1        FREE PTRLIST; 100
64 1 1        END; 101
65 1 1        PUT SKIP LIST ("END CYCLES ANALYSIS"); 102
66 1          /*.....*/ 103
67 1          /* 104
        105
        106
        107
        108
        109
        110
        111
        112
        113
        114
        115
        116
        117
        118
        119
        120
        121
        122
        123
        124
        125
        126
        127
        128
        129
        130
        131
        132
        133
        134
        135
        136
        137
        138
        139
        140
        141
        142
        143
        144
        145
        146
        147
        148
        149
        150
        151
        152
        153
        154
        155
        156
        157
        158
        159
        160
        161
        162
        163
        164
        165
        166
        167
        168
        169
        170
        171
        172
        173
        174
        175
        176
        177
        178
        179
        180
        181
        182
        183
        184
        185
        186
        187
        188
        189
        190
        191
        192
        193
        194
        195
        196
        197
        198
        199
        200
        201
        202
        203
        204
        205
        206
        207
        208
        209
        210
        211
        212
        213
        214
        215
        216
        217
        218
        219
        220
        221
        222
        223
        224
        225
        226
        227
        228
        229
        230
        231
        232
        233
        234
        235
        236
        237
        238
        239
        240
        241
        242
        243
        244
        245
        246
        247
        248
        249
        250
        251
        252
        253
        254
        255
        256
        257
        258
        259
        260
        261
        262
        263
        264
        265
        266
        267
        268
        269
        270
        271
        272
        273
        274
        275
        276
        277
        278
        279
        280
        281
        282
        283
        284
        285
        286
        287
        288
        289
        290
        291
        292
        293
        294
        295
        296
        297
        298
        299
        300
        301
        302
        303
        304
        305
        306
        307
        308
        309
        310
        311
        312
        313
        314
        315
        316
        317
        318
        319
        320
        321
        322
        323
        324
        325
        326
        327
        328
        329
        330
        331
        332
        333
        334
        335
        336
        337
        338
        339
        340
        341
        342
        343
        344
        345
        346
        347
        348
        349
        350
        351
        352
        353
        354
        355
        356
        357
        358
        359
        360
        361
        362
        363
        364
        365
        366
        367
        368
        369
        370
        371
        372
        373
        374
        375
        376
        377
        378
        379
        380
        381
        382
        383
        384
        385
        386
        387
        388
        389
        390
        391
        392
        393
        394
        395
        396
        397
        398
        399
        400
        401
        402
        403
        404
        405
        406
        407
        408
        409
        410
        411
        412
        413
        414
        415
        416
        417
        418
        419
        420
        421
        422
        423
        424
        425
        426
        427
        428
        429
        430
        431
        432
        433
        434
        435
        436
        437
        438
        439
        440
        441
        442
        443
        444
        445
        446
        447
        448
        449
        450
        451
        452
        453
        454
        455
        456
        457
        458
        459
        460
        461
        462
        463
        464
        465
        466
        467
        468
        469
        470
        471
        472
        473
        474
        475
        476
        477
        478
        479
        480
        481
        482
        483
        484
        485
        486
        487
        488
        489
        490
        491
        492
        493
        494
        495
        496
        497
        498
        499
        500
        501
        502
        503
        504
        505
        506
        507
        508
        509
        510
        511
        512
        513
        514
        515
        516
        517
        518
        519
        520
        521
        522
        523
        524
        525
        526
        527
        528
        529
        530
        531
        532
        533
        534
        535
        536
        537
        538
        539
        540
        541
        542
        543
        544
        545
        546
        547
        548
        549
        550
        551
        552
        553
        554
        555
        556
        557
        558
        559
        560
        561
        562
        563
        564
        565
        566
        567
        568
        569
        570
        571
        572
        573
        574
        575
        576
        577
        578
        579
        580
        581
        582
        583
        584
        585
        586
        587
        588
        589
        590
        591
        592
        593
        594
        595
        596
        597
        598
        599
        600
        601
        602
        603
        604
        605
        606
        607
        608
        609
        610
        611
        612
        613
        614
        615
        616
        617
        618
        619
        620
        621
        622
        623
        624
        625
        626
        627
        628
        629
        630
        631
        632
        633
        634
        635
        636
        637
        638
        639
        640
        641
        642
        643
        644
        645
        646
        647
        648
        649
        650
        651
        652
        653
        654
        655
        656
        657
        658
        659
        660
        661
        662
        663
        664
        665
        666
        667
        668
        669
        670
        671
        672
        673
        674
        675
        676
        677
        678
        679
        680
        681
        682
        683
        684
        685
        686
        687
        688
        689
        690
        691
        692
        693
        694
        695
        696
        697
        698
        699
        700
        701
        702
        703
        704
        705
        706
        707
        708
        709
        710
        711
        712
        713
        714
        715
        716
        717
        718
        719
        720
        721
        722
        723
        724
        725
        726
        727
        728
        729
        730
        731
        732
        733
        734
        735
        736
        737
        738
        739
        740
        741
        742
        743
        744
        745
        746
        747
        748
        749
        750
        751
        752
        753
        754
        755
        756
        757
        758
        759
        760
        761
        762
        763
        764
        765
        766
        767
        768
        769
        770
        771
        772
        773
        774
        775
        776
        777
        778
        779
        780
        781
        782
        783
        784
        785
        786
        787
        788
        789
        790
        791
        792
        793
        794
        795
        796
        797
        798
        799
        800
        801
        802
        803
        804
        805
        806
        807
        808
        809
        810
        811
        812
        813
        814
        815
        816
        817
        818
        819
        820
        821
        822
        823
        824
        825
        826
        827
        828
        829
        830
        831
        832
        833
        834
        835
        836
        837
        838
        839
        840
        841
        842
        843
        844
        845
        846
        847
        848
        849
        850
        851
        852
        853
        854
        855
        856
        857
        858
        859
        860
        861
        862
        863
        864
        865
        866
        867
        868
        869
        870
        871
        872
        873
        874
        875
        876
        877
        878
        879
        880
        881
        882
        883
        884
        885
        886
        887
        888
        889
        890
        891
        892
        893
        894
        895
        896
        897
        898
        899
        900
        901
        902
        903
        904
        905
        906
        907
        908
        909
        910
        911
        912
        913
        914
        915
        916
        917
        918
        919
        920
        921
        922
        923
        924
        925
        926
        927
        928
        929
        930
        931
        932
        933
        934
        935
        936
        937
        938
        939
        940
        941
        942
        943
        944
        945
        946
        947
        948
        949
        950
        951
        952
        953
        954
        955
        956
        957
        958
        959
        960
        961
        962
        963
        964
        965
        966
        967
        968
        969
        970
        971
        972
        973
        974
        975
        976
        977
        978
        979
        980
        981
        982
        983
        984
        985
        986
        987
        988
        989
        990
        991
        992
        993
        994
        995
        996
        997
        998
        999
        1000
        1001
        1002
        1003
        1004
        1005
        1006
        1007
        1008
        1009
        1010
        1011
        1012
        1013
        1014
        1015
        1016
        1017
        1018
        1019
        1020
        1021
        1022
        1023
        1024
        1025
        1026
        1027
        1028
        1029
        1030
        1031
        1032
        1033
        1034
        1035
        1036
        1037
        1038
        1039
        1040
        1041
        1042
        1043
        1044
        1045
        1046
        1047
        1048
        1049
        1050
        1051
        1052
        1053
        1054
        1055
        1056
        1057
        1058
        1059
        1060
        1061
        1062
        1063
        1064
        1065
        1066
        1067
        1068
        1069
        1070
        1071
        1072
        1073
        1074
        1075
        1076
        1077
        1078
        1079
        1080
        1081
        1082
        1083
        1084
        1085
        1086
        1087
        1088
        1089
        1090
        1091
        1092
        1093
        1094
        1095
        1096
        1097
        1098
        1099
        1100
        1101
        1102
        1103
        1104
        1105
        1106
        1107
        1108
        1109
        1110
        1111
        1112
        1113
        1114
        1115
        1116
        1117
        1118
        1119
        1120
        1121
        1122
        1123
        1124
        1125
        1126
        1127
        1128
        1129
        1130
        1131
        1132
        1133
        1134
        1135
        1136
        1137
        1138
        1139
        1140
        1141
        1142
        1143
        1144
        1145
        1146
        1147
        1148
        1149
        1150
        1151
        1152
        1153
        1154
        1155
        1156
        1157
        1158
        1159
        1160
        1161
        1162
        1163
        1164
        1165
        1166
        1167
        1168
        1169
        1170
        1171
        1172
        1173
        1174
        1175
        1176
        1177
        1178
        1179
        1180
        1181
        1182
        1183
        1184
        1185
        1186
        1187
        1188
        1189
        1190
        1191
        1192
        1193
        1194
        1195
        1196
        1197
        1198
        1199
        1200
        1201
        1202
        1203
        1204
        1205
        1206
        1207
        1208
        1209
        1210
        1211
        1212
        1213
        1214
        1215
        1216
        1217
        1218
        1219
        1220
        1221
        1222
        1223
        1224
        1225
        1226
        1227
        1228
        1229
        1230
        1231
        1232
        1233
        1234
        1235
        1236
        1237
        1238
        1239
        1240
        1241
        1242
        1243
        1244
        1245
        1246
        1247
        1248
        1249
        1250
        1251
        1252
        1253
        1254
        1255
        1256
        1257
        1258
        1259
        1260
        1261
        1262
        1263
        1264
        1265
        1266
        1267
        1268
        1269
        1270
        1271
        1272
        1273
        1274
        1275
        1276
        1277
        1278
        1279
        1280
        1281
        1282
        1283
        1284
        1285
        1286
        1287
        1288
        1289
        1290
        1291
        1292
        1293
        1294
        1295
        1296
        1297
        1298
        1299
        1300
        1301
        1302
        1303
        1304
        1305
        1306
        1307
        1308
        1309
        1310
        1311
        1312
        1313
        1314
        1315
        1316
        1317
        1318
        1319
        1320
        1321
        1322
        1323
        1324
        1325
        1326
        1327
        1328
        1329
        1330
        1331
        1332
        1333
        1334
        1335
        1336
        1337
        1338
        1339
        1340
        1341
        1342
        1343
        1344
        1345
        1346
        1347
        1348
        1349
        1350
        1351
        1352
        1353
        1354
        1355
        1356
        1357
        1358
        1359
        1360
        1361
        1362
        1363
        1364
        1365
        1366
        1367
        1368
        1369
        1370
        1371
        1372
        1373
        1374
        1375
        1376
        1377
        1378
        1379
        1380
        1381
        1382
        1383
        1384
        1385
        1386
        1387
        1388
        1389
        1390
        1391
        1392
        1393
        1394
        1395
        1396
        1397
        1398
        1399
        1400
        1401
        1402
        1403
        1404
        1405
        1406
        1407
        1408
        1409
        1410
        1411
        1412
        1413
        1414
        1415
        1416
        1417
        1418
        1419
        1420
        1421
        1422
        1423
        1424
        1425
        1426
        1427
        1428
        1429
        1430
        1431
        1432
        1433
        1434
        1435
        1436
        1437
        1438
        1439
        1440
        1441
        1442
        1443
        1444
        1445
        1446
        1447
        1448
        1449
        1450
        1451
        1452
        1453
        1454
        1455
        1456
        1457
        1458
        1459
        1460
        1461
        1462
        1463
        1464
        1465
        1466
        1467
        1468
        1469
        1470
        1471
        1472
        1473
        1474
        1475
        1476
        1477
        1478
        1479
        1480
        1481
        1482
        1483
        1484
        1485
        1486
        1487
        1488
        1489
        1490
        1491
        1492
        1493
        1494
        1495
        1496
        1497
        1498
        1499
        1500
        1501
        1502
        1503
        1504
        1505
        1506
        1507
        1508
        1509
        1510
        1511
        1512
        1513
        1514
        1515
        1516
        1517
        1518
        1519
        1520
        1521
        1522
        1523
        1524
        1525
        1526
        1527
        1528
        1529
        1530
        1531
        1532
        1533
        1534
        1535
        1536
        1537
        1538
        1539
        1540
        1541
        1542
        1543
        1544
        1545
        1546
        1547
        1548
        1549
        1550
        1551
        1552
        1553
        1554
        1555
        1556
        1557
        1558
        1559
        1560
        1561
        1562
        1563
        1564
        1565
        1566
        1567
        1568
        1569
        1570
        1571
        1572
        1573
        1574
        1575
        1576
        1577
        1578
        1579
        1580
        1581
        1582
        1583
        1584
        1585
        1586
        1587
        1588
        1589
        1590
        1591
        1592
        1593
        1594
        1595
        1596
        1597
        1598
        1599
        1600
        1601
        1602
        1603
        1604
        1605
        1606
        1607
        1608
        1609
        1610
        1611
        1612
        1613
        1614
        1615
        1616
        1617
        1618
        1619
        1620
        1621
        1622
        1623
        1624
        1625
        1626
        1627
        1628
        1629
        1630
        1631
        1632
        1633
        1634
        1635
        1636
        1637
        1638
        1639
        1640
        1641
        1642
        1643
        1644
        1645
        1646
        1647
        1648
        1649
        1650
        1651
        1652
        1653
        1654
        1655
        1656
        1657
        1658
        1659
        1660
        1661
        1662
        1663
        1664
        1665
        1666
        1667
        1668
        1669
        1670
        1671
        1672
        1673
        1674
        1675
        1676
        1677
        1678
        1679
        1680
        1681
        1682
        1683
        1684
        1685
        1686
        1687
        1688
        1689
        1690
        1691
        1692
        1693
        1694
        1695
        1696
        1697
        1698
        1699
        1700
        1701
        1702
        1703
        1704
        1705
        1706
        1707
        1708
        1709
        1710
        1711
        1712
        1713
        1714
        1715
        1716
        1717
        1718
        1719
        1720
        1721
        1722
        1723
        1724
        1725
        1726
        1727
        1728
        1729
        1730
        1731
        1732
        1733
        1734
        1735
        1736
        1737
        1738
        1739
        1740
        1741
        1742
        1743
        1744
        1745
        1746
        1747
        1748
        1749
        1750
        1751
        1752
        1753
        1754
        1755
        1756
        1757
        1758
        1759
        1760
        1761
        1762
        1763
        1764
        1765
        1766
        1767
        1768
        1769
        1770
        1771
        1772
        1773
        1774
        1775
        1776
        1777
        1778
        1779
        1780
        1781
        1782
        1783
        1784
        1785
        1786
        1787
        1788
        1789
        1790
        1791
        1792
        1793
        1794
        1795
        1796
        1797
        1798
        1799
        1800
        1801
        1802
        1803
        1804
        1805
        1806
        1807
        1808
        1809
        1810
        1811
        1812
        1813
        1814
        1815
        1816
        1817
        1818
        1819
        1820
        1821
        1822
        1823
        1824
        1825
        1826
        1827
        1828
        1829
        1830
        1831
        1832
        1833
        1834
        1835
        1836
        1837
        1838
        1839
        1840
        1841
        1842
        1843
        1844
        1845
        1846
        1847
        1848
        1849
        1850
        1851
        1852
        1853
        1854
        1855
        1856
        1857
        1858
        1859
        1860
        1861
        1862
        1863
        1864
        1865
        1866
        1867
        1868
        1869
        1870
        1871
        1872
        1873
        1874
        1875
        1876
        1877
        1878
        1879
        1880
        1881
        1882
        1883
        1884
```

```

/* THE POINTER TO THE ADJACENCY LIST STRUCTURE "ADJ"
   IS GIVEN BY PROCEDURE "ACRDPN". THE STRUCTURE "ADJ"
   GIVES THE PICTURE OF THE INCIDENT EDGES AND ADJACENT
   NODES OF THE CURRENT VERTEX.*/

/* DEFINITIONS:
   "LOWLINK(V)" OF A VERTEX V IS THE SMALLEST VERTEX
   WHICH IS IN THE SAME COMPONENT AS V AND IS REACHABLE
   BY TRAVERSING ZERO OR MORE TREE ARCS FOLLOWED BY AT
   MOST ONE "BRAND" OR "CROSS-LINK" (AN ARC TO A VERTEX
   ALREADY REACHED).

   "V" IS THE ROOT OF A COMPONENT IFF LOWLINK(V)=V */

/* THE FOLLOWING ARRAYS ARE MAINTAINED IN THIS
   PROCEDURE FOR THE REASONS MENTIONED BELOW :
   L O W L I N K : TO KEEP TRACK OF LOWLINK VALUES OF
                   EACH VERTEX
   N U M B       : TO NUMBER EACH VERTEX AS IT IS TRACED
   S T A C K     : TO PUSH THE VERTICES ALREADY TRACED ON
                   A STACK
   GNSTACK       : TO VERIFY A VERTEX IS PUSHED ON S T A C K
   /* PTR IS A POINTER USED TO KEEP TRACK OF ELEMENTS
      IN S T A C K */

   /* "COMP" IS A STRUCTURE USED TO KEEP THE ELEMENTS
      WHICH ARE STRONGLY-CONNECTED AS A LINKED LIST */

```

WANA 7963 05/05/78 PLIC-VER-033078

PAGE

STPT LEVEL NEST

73	2		DCL (P,FIRST,LAST,L,ADJMP) PTR;	152
74	2		DCL V BIN FIXED;	153
75	2		DCL (J,W) BIN FIXED;	154
76	2		DCL (P,CLOWE) BIN FIXED;	155
77	2		CALL ACNDRA(V,ADJMP);	156
78	2		LOWLINK(V),NUMB(V),I=1+1;	157
			/* PUT V ON STACK OF POINTS */	158
79	2		CALL ADL (V,STACK,PTR);	159
80	2		ONSTACK(V) = "1"B;	160
81	2		IF (ADJMP=NULL) THEN GO TO NOADJ;	161
83	2		IF (CRUSED=0) THEN GO TO NOADJ;	162
85	2		DO J=1 TO RUDED;	163
86	2	1	W=ENTRIES(J).COL;	164
87	2	1	IF (W=0) THEN GO TO SAM;	165
			/* W IS IN THE ADJACENCY LIST OF V */	166
			IF (NUMB(W)=0) /* (V,W) IS A TREE ARC */	167
			THEN DO;	168
89	2	1	CALL STRGCON (W,W);	169
90	2	1	LOWLINK(V)=MIN(LOWLINK(V),LOWLINK(W));	170
91	2	2	END;	171
92	2	2	ELSE IF ((NUMB(W)<NUMB(V))&ONSTACK(W))	172
93	2	2	/* (V,W) IS A FRONT OR CROSS LINK */	173
94	2	1	THEN LOWLINK(V)=MIN(LOWLINK(V),NUMB(W));	174
95	2	1	SAM: END;	175
96	2	1	NOADJ: IF (LOWLINK(V)=NUMB(V))	176
97	2		THEN DO;	177
98	2		FIRST=NULL;	178
99	2	1	/* V IS THE ROOT OF THE COMPONENT */	179
			/* START A NEW STRONGLY CONNECTED COMPONENT */	180
100	2	1	COUNT=COUNT+1;	181
101	2	1	DO WHILE (NUMB(STACK(PTR))=NUMB(V)&(PTR>0));	182
			/*DELETE THE POINT FROM TOP OF STACK AND PUT	183
			IT IN THE CURRENT COMPONENT */	184
102	2	2	UNSTACK(STACK(PTR))="0"B;	185
103	2	2	ONCPF(STACK(PTR))="1"B;	186
104	2	2	ANAP(STACK(PTR))=COUNT;	187
105	2	2	ALLOCATE CUMP;	188
106	2	2	IF FIRST=NULL THEN FIRST=P;	189
107	2	2	ELSE LAST->LINK=P;	190
108	2	2	P->INDEX=STACK(PTR);	191
109	2	2	P->LINK=NULL;	192
110	2	2	P->LIST=NULL;	193
111	2	2	LAST=P;	194
112	2	2	CALL DELETE (STACK,PTR);	195
113	2	4	END;	196
114	2	4	IF (FIRST->LINK=NULL) THEN DO;	197
115	2	1	IF ("SUB) THEN CALL ANALCMP;	198
116	2	2	IF (PCOMP=NULL) THEN ALLOCATE PTRLIST;	199
117	2	2	IF SUB THEN CMPLD=0;	200
118	2	2	CALL PRECMP;	201

GANA 7967 05/05/78 PLIC-VLR-033078

PAGE

STMT LEVEL NEST

```

124 2 2      PIRCOMP=GROUPS=FIRST; 202
125 2 2      IF (CMPCODE="S" OR "SUB") THEN CALL LINKAS; 203
127 2 4      END; 204
128 2 1      ELSE FREE COMP; 205
129 2 1      END; 206
130 2      ONCMP="0"; 207
/* INTERNAL PROCEDURES */ 208
131 2  PROCEDURE (V,STACK,PTR) ; 209
132 3  DCL (V,STACK(*),PTR) FIXED BIN ; 210
133 3  PIR = PTR+1; 211
134 3  STACK(PTR) = V; 212
135 3  END PROC; 213
136 2  DELETE: PROCEDURE (STACK,PTR); 214
137 3  DCL (STACK(*),PTR) FIXED BIN ; 215
138 3  STACK(PTR) = 0; 216
139 3  PIR = PTR-1; 217
140 3  END DELETE; 218
/*.....*/ 219
/*      PRINTING ROUTINE      */ 220
/*.....*/ 221
/*.....*/ 222
/*.....*/ 223
141 2  PCOMP: PROC ; 224
142 3  DCL 1 STORAGE_ENTRY BASED(STORAGE_PTR), 225
      2 KEYS FIXED BIN, 226
      2 DATA_PT POINTER, 227
      2 KEY_ENTRY(N KEYS), 228
      3 NAMEPTR POINTER, 229
      3 NEXT_PTR POINTER; 230
143 3  DCL 1 DIRECTORY_ENTRY BASED(DIRECTORY_PTR), 231
      2 KEY_NAME CHAR( 10 ), 232
      2 UP_PTR POINTER, 233
      2 DOWN_PTR POINTER, 234
      2 FIRST_IN_LIST POINTER; 235
144 3  DCL REEL-OF-ENTRY (LCHAR(*) VAR) 236
      RETURNS (CHAR(31) VAR); 237
145 3  DCL PRINT_LINE CHAR(133) BASED(PRL); 238
146 3  DCL PREFIX CHAR(2); 239
147 3  DCL NAME CHAR( 10 -2) VAR; 240
148 3  PUT SKIP LIST ("ENTERING PCOMP"); 241
149 3  IF FIRST_TIME THEN DO; 242
151 3 1  FIRST_TIME="0"; 243
152 3 1  #GROUPS=1; 244
153 3 1  LOCATE PRINT_LINE FILE (ADJMPRT); 245
154 3 1  PRINT_LINE="1 THE FOLLOWING GROUPS OF ITEMS ARE CIRCULARLY DESCRIBED"; 246
155 3 1  LOCATE PRINT_LINE FILE (ADJMPRT); 247
156 3 1  PRINT_LINE=" ( DICTN , PREFIX CODE , NAME )"; 248
157 3 1  LOCATE PRINT_LINE FILE (ADJMPRT); 249
158 3 1  PRINT_LINE="0"; 250
159 3 1  END; 251
160 3  ELSE #GROUPS=#GROUPS+1; 252

```

GANA 7963 05/05/78 PLIC-VER-033078

PAGE

STMT LEVEL NEST

```

161 3 IF CMPCODE = 1 THEN DO; 243
162 3 1 LOCATE PRINT_LINE FILE(ADJMRPT); 244
164 3 1 PRINT_LINE=" ***ERROR 1 CYCLES*** FOLLOWING LOOP MUST BE OPENED"; 245
165 3 1 LOCATE PRINT_LINE FILE(ADJMRPT); 246
166 3 1 PRINT_LINE=" ELEMENTS IN THE LOOP OTHER THAN ASSERTION OR DATA ITEM 247
      S" 247
      ; 248
167 3 1 END; 249
168 3 ELSE 250
169 3 IF CMPCODE = 2 THEN DO; 251
170 3 1 LOCATE PRINT_LINE FILE(ADJMRPT); 252
171 3 1 PRINT_LINE=" ***ERROR 2 CYCLES*** FOLLOWING LOOP INCONSISTENT"; 253
172 3 1 LOCATE PRINT_LINE FILE(ADJMRPT); 254
173 3 1 PRINT_LINE=" INCONSISTENT REFERENCES BETWEEN ASSERTIONS AND DATA NA 255
      MES" 255
      ; 256
174 3 1 END; 257
175 3 ELSE 258
176 3 IF CMPCODE = 3 THEN DO; 259
177 3 1 LOCATE PRINT_LINE FILE(ADJMRPT); 260
178 3 1 PRINT_LINE=" ***ERROR 3 CYCLES*** IMPROPER NORMALIZATION"; 261
179 3 1 LOCATE PRINT_LINE FILE(ADJMRPT); 262
180 3 1 PRINT_LINE=" FOLLOWING GROUP CONTAINS SINGLE ASSERTION WITH MULTIPL 263
      F TARGET NAMES" 263
      ; 264
181 3 1 END; 265
182 3 P=FIRST; 266
183 3 DO WHILE (P=NULL); 267
184 3 1 CALL ACPDR(INDEX, STORAGE_PTR); 268
185 3 1 DIRECTORY_PTR=NAMEPTR(2); 269
186 3 1 PREFIX=SUBSTR(KEY_NAME,1,2); 270
187 3 1 NAME=SUBSTR(KEY_NAME,3); 271
188 3 1 LOCATE PRINT_LINE FILE(ADJMRPT); 272
189 3 1 PRINT_LINE=" ||INDEX|| " ||PREFIX|| " ||RECLBUF(NAME); 273
190 3 1 P=P->LINK; 274
191 3 1 END; 275
192 3 LOCATE PRINT_LINE FILE(ADJMRPT); 276
193 3 PRINT_LINE=" ||(32)"-"; 277
194 3 LOCATE PRINT_LINE FILE(ADJMRPT); 278
195 3 PRINT_LINE=" "; 279
196 3 END PRCOMP; 280
/* ***** 281
/* 282
/* PROCEDURE FOR ANALYSIS OF STRONGLY CONNECTED COMPONENTS */ 283
/* 284
/* ***** 285
197 2 ANALCMP= PROC ; 286
198 3 /* ANALYZE STRONGLY CONNECTED COMPONENTS */ 287
199 3 DCL ULT PTR; 288
200 3 CMPCODE=0; 289
200 3 P=FIRST; 290

```

GANA 7963 05/05/78 PLIC-VER-033078

PAGE

STPT LEVEL NEST

```

201      3      /*CREATE ADJACENCY LIST FOR STRONGLY CONNECTED COMPONENTS*/ 291
202      3      DO WHILE (P=NULL); 292
203      3      1      CALL ACROPA(P->INDEX,ADJMP); 293
204      3      1      DO J=1 TO NUSED; 294
205      3      2      IF CACNP(ENTRIES.COL(J)) THEN DO; 295
206      3      3      ALLOCATE CPLIST; 296
207      3      3      IF P->LIST=NULL THEN P->LIST=L; 297
208      3      3      ELSE UL->NEXT=L; 298
209      3      3      L->ALJNODE=ENTRIES.COL(J); 299
210      3      3      L->TYPE=ENTRIES.CVALUE(J); 300
211      3      3      L->NEXT=NULL; 301
212      3      3      UL=L; 302
213      3      3      END; 303
214      3      3      END; 304
215      3      2      END; 305
216      3      1      /*CHECK CONSISTENCY OF PRECEDENCE RELATIONSHIPS*/ 306
217      3      1      L=P->LIST; 307
218      3      1      IF (L->TYPE=3) THEN 308
219      4      1      BEGIN; 309
220      4      2      DO WHILE (L->NEXT=NULL & CMPCODE=0); 310
221      4      3      L=L->NEXT; 311
222      4      3      IF (L->TYPE=3) THEN DO; 312
223      4      4      IF (L->TYPE=4) THEN CMPCODE=2; 313
224      4      4      ELSE IF (L->TYPE=8) THEN CMPCODE = 1; 314
225      4      4      END; 315
226      4      3      END; 316
227      4      3      IF CMPCODE=0 THEN P->KEY="SV"; 317
228      4      3      ELSE P->KEY="SE"; 318
229      4      3      END; 319
230      3      1      ELSE 320
231      3      1      IF (L->TYPE=8) THEN 321
232      4      1      BEGIN; 322
233      4      2      DO WHILE (L->NEXT=NULL & CMPCODE=0); 323
234      4      3      L=L->NEXT; 324
235      4      3      IF (L->TYPE=8) THEN DO; 325
236      4      4      IF (L->TYPE=4) THEN CMPCODE=2; 326
237      4      4      ELSE IF (L->TYPE=3) THEN CMPCODE=1; 327
238      4      4      END; 328
239      4      3      END; 329
240      3      1      IF CMPCODE=0 THEN P->KEY="SV"; 330
241      3      1      ELSE P->KEY="SE"; 331
242      3      1      END; 332
243      3      1      ELSE 333
244      3      1      IF (L->TYPE=4) THEN DO; 334
245      3      2      IF (L->NEXT=NULL) THEN 335
246      4      2      BEGIN; 336
247      4      3      DO WHILE (L->NEXT=NULL & CMPCODE=0); 337
248      4      4      L=L->NEXT; 338
249      4      4      IF (L->TYPE=4) THEN DO; 339
250      4      5      IF (L->TYPE=3) THEN CMPCODE=2; 340
251      4      5      ELSE CMPCODE=1; 341
252      4      5      END; 342
253      4      4      END; 343
254      4      3      END; 344
255      3      1      END; 345
256      3      1      END; 346
257      3      1      END; 347
258      3      1      END; 348
259      3      1      END; 349
260      3      1      END; 350

```

GANA 7963 05/05/78 PLIC-VER-033078

PAGE

STMT LEVEL NEST

261	4	3	END;	342
			/*CHECK IF ASSERTION WITH MULTIPLE TARGETS WAS "MERGED" BEFORE*/	343
262	4	2	IF CMPCODE=0 THEN CALL CHKPLT;	344
264	4	2	END;	345
265	3	2	IF CMPCODE=0 THEN P->KEY="SA";	346
267	3	2	ELSE P->KEY="SE";	347
268	3	2	END;	348
269	3	1	ELSE DO;	349
270	3	2	CMPCODE=1;	350
271	3	2	P->KEY="SE";	351
272	3	2	END;	352
273	3	1	P=P->LINK;	353
274	3	1	END;	354
275	3		END ANALCMP;	355
			/*.....*/	356
			/*	357
			/* "LINKING" STORAGE ENTRIES OF S-C-C IN THE AM	358
			/*	359
			/*.....*/	360
276	2		LINKAS: PROC;	361
277	3		DCL 1 STORAGE_ENTRY BASED (STORAGE_PTR),	637
			2 KEYS FIXED BIN,	638
			2 DATA_PTR POINTER,	639
			2 KEY_ENTRY (N REFER (NKEYS)),	640
			3 NAMEPTR POINTER,	641
			3 NEXT_PTR POINTER;	642
			/* DATA LIST OF AN ASSERTION */	643
278	3		DCL 1 ASS_ST_DATA BASED (DATA_PTR),	644
			2 COMMENT_PTR PTR, /*POINTER TO COMMENT STACK*/	645
			2 USER_LINE BIN FIXED (31,0), /*USER SUPPLIED LINE */	660
			2 PLINL BIN FIXED (31,0), /*PROCESSOR SUPPLIED LINE */	661
			2 DATA_LIST_PTR, /*POINTER TO ADDITIONAL DATA*/	662
			2 MAPPING_LIN FIXED (31,0), /*NOT USED*/	663
			2 STVLST_PTR PTR, /*POINTER TO SOURCE & TARGET VARIABLE LIST*/	646
			2 ASS_PTR PTR, /*POINTER TO ASSERTION ITSELF*/	647
			2 NASS_PTR PTR, /*POINTER TO NEXT ASSERTION	648
			IF PART OF SIMULTANEOUS SET*/	648
				649
			2 SV_PTR PTR, /*POINTER TO SOURCE	650
			VARIABLE LIST*/	650
				651
			2 TV_PTR PTR, /*POINTER TO TARGET	652
			VARIABLE LIST*/	652
				653
			2 FN_PTR PTR, /*POINTER TO FUNCTION LIST*/	654
			2 INIT_PTR PTR, /*POINTER TO INITIAL	655
			VALUE LIST*/	655
				656
			2 TEST_PTR PTR, /*POINTER TO TEST VALUE LIST*/	657
			2 SOL_PTR PTR, /*POINTER TO SOLUTION INDICATOR*/	658
279	3		DCL 1 NASS_LIST BASED (NASSPTR),	664

GANA 7963 US/CS/78 PLIC-VER-033678

PAGE 1

SYMT LEVEL NEST

```

2 CODE CHAR(1), /* "D" IF NEXT_PTR POINTS TO DATA AREA,
  "S", IF NEXT_PTR POINTS TO STORAGE ENTRY */
2 FIRST_PTR PTR, /* POINTER TO THE FIRST STORAGE ENTRY */
2 NEXT_PTR PTR, /* POINTS TO THE NEXT DATA AREA
  OR STORAGE ENTRY */
280 3   DCL 1 ERROR_DESC BASED(DATA_PTR),
        2 PADDING (4) BIN FIXED(21,0),
        2 DELETED BIN FIXED,
        2 ORTOP BIN FIXED;
281 3   IF FLAGLNK THEN RETURN;
283 3   P=FIRST;
284 3   NASSPTR=NULL;
285 3   FIRST=NULL;
286 3   DO WHILE (P=NULL);
287 3     1 IF (KEY="SA") THEN DO;
289 3       2 CALL ACORDR(INDEX,STORAGE_PTR);
        /*
        *
        * SET CODE IN ERROR# FOR LATER PRINTING OF
        * SIMULTANEOUS ASSESSMENTS
        *
        */
290 3     2 DATA_PTR = DATA_PT;
291 3     2 ERROR# = 38;
        /*
        *
        */
292 3     2 IF NASSPTR=NULL THEN NASS_LIST.NEXT_PTR=STORAGE_PTR;
294 3     2 DATA_PTR=DATA_PT;
295 3     2 IF (NASS_PTR=NULL) THEN DO;
297 3       3 ALLOCATE NASS_LIST;
298 3       3 NASS_PTR=NASSPTR;
299 3     3 END;
300 3     2 ELSE DO;
301 3       3 NASSPTR=NASS_PTR;
302 3       3 CALL GETLAST;
303 3     3 END;
304 3     2 CODE="S";
305 3     2 FIRST_PTR=FIRST;
306 3     2 END;
307 3     1 P=LINK;
308 3     1 END;
309 3     NASS_LIST.NEXT_PTR=NULL;
310 3     CODE=" ";
311 3   CITLAST: PROCEDURE RECURSIVE;
        /* INTERNAL RECURSIVE PROC USED TO OBTAIN LAST NASS-PTR
        IN A LIST OF DATA AREAS AND STORAGE ENTRIES FOR
        LINKAGE WITH NEXT STORAGE ENTRY */

```

GANA 7963 05/05/78 PLIC-VER-033078

PAGE

SYMT LEVEL NEST

```

312 4      IF CODE=" " THEN RETURN; 404
314 4      IF CODE="D" THEN NASSPTR=NASS_LIST.NEXT_PTR; 405
316 4      ELSE IF CODE="S" THEN DO; 406
318 4      1      STORAGE_PTR=NASS_LIST.NEXT_PTR; 407
319 4      1      DATA_PTR=DATA_PT; 408
320 4      1      NASSPTR=NASS_PTR; 409
321 4      1      END; 410
322 4      ELSE CODE=" "; 411
323 4      CALL GETLAST; 412
324 4      END GETLAST; 413
325 3      END LINKAS; 414
      /*.....*/ 415
      /* 416
      /* CHECKS ASSERTIONS WITH MORE THAN ONE TARGET VARIABLE 417
      /* 418
      /*.....*/ 419
326 2      CHKMULT: PROC; 420
      /*CHECKS THAT ASSERTIONS WITH MORE THAN ONE TARGET ARE 421
      PROPERLY MERGED */ 422
327 3      CMPCODE=0; 423
328 3      END CHKMULT; 424
329 2      END STRGCON ; 425
      /*.....*/ 426
      /* 427
      /* PRINT SIMULTANEOUS ASSERTIONS TO REPORT 428
      /* 429
      /*.....*/ 430
330 1      PSIMASS: PROCEDURE; 431
331 2      DCL 1 STORAGE_ENTRY BASED(STORAGE_PTR), 432
          2 KEYS FIXED BIN, 433
          2 DATA_PT POINTER, 434
          2 KEY_ENTRY(N REFER(4KEYS)), 435
          2 NAMEPTR POINTER, 436
          2 NEXT_PTR POINTER; 437
332 2      DCL PRTHASST ENTRY (POINTER); 438
333 2      DCL PRTHCK ENTRY; 439
334 2      DCL CENTERL ENTRY (CHAR(4) VAR); 440
335 2      DCL PRTRL ENTRY; 441
336 2      DCL #COMP BIN FIXED; 442
337 2      #COMP=4GROUPS; 443
338 2      DO WHILE (#COMP > 0); 444
339 2      1      FIRST = #TRCMP(#COMP); 445
340 2      1      CALL PRTHCK; 446
341 2      1      CALL CENTERL ("SIMULTANEOUS ASSERTIONS"); 447
342 2      1      CALL PRTRL; 448
343 2      1      DO WHILE (FIRST /= NULL);
344 2      2      P=FIRST;
345 2      2      IF KEY="SA" THEN DO;
346 2      3      CALL ACORR (INDEX,STORAGE_PTR);

```

LANA 7063 05/05/78 PLIC-VLR-033078

PAGE

STRT LEVEL NEST

349	2	CALL PHTASS1 (STORAGE_PTR);	467
350	2	END;	480
351	2	FIRST=LINK;	481
352	2	END;	482
353	2	CALL PRTHCN;	483
354	2	CALL CINTRL ("END OF SIMULTANEOUS GROUP");	484
355	2	CALL PHTPL;	485
356	2	ACOMP=PCOMP-1;	486
357	2	END;	487
		END PSIMASS;	488
		/*****	489
		/*	490
		RANKING NODES OF DIGRAPH	491
		/*	492
		*****/	493
		/*	494
		**	495
		**	496
		TEMPORARY PROCEDURE, SHOULD BE CHANGED	497
		TO LIST PROCESSING.	498
		**	499
		**	500
		*/	501
358	1	PRECED: PROCEDURE(N);	502
359	2	DCL A(NN,N) LIT(1) CTL;	503
360	2	DCL (I,J,M) FIXED BIN;	504
		/* CREATE REDUCED ACYCLIC BIT ADJACENCY MATRIX */	505
361	2	IF FLAGANK THEN RETURN;	506
362	2	NN = COUNT;	507
363	2	ALLOCATE A;	508
364	2	A = "0B";	509
365	2	DO I=1 TO N;	510
366	2	CALL ACKDRA(I,ADJMP);	511
367	2	IF (KUSED=0) THEN GO TO NOADJ;	512
368	2	GO J=1 TO KUSED;	513
369	2	M=ENTRIES(J).COL;	514
370	2	IF (M=0) THEN GO TO OUT;	515
371	2	IF (A*AP(I)=A*MAP(M)) THEN	516
372	2	A(A*AP(I),A*AP(M))="1B";	517
373	2	OUT: END;	518
374	2	Noadj: END;	519
375	2	CALL RANKMOD(A,NN,N);	520
376	2	FREE A;	521
377	2	PLOT SKIP LIST ("ORDER VECTOR:");	522
378	2	PLOT SKIP EDIT (ORDER) ("F(S)");	523
379	2	RANKMOD: PROCEDURE(A,NN,N);	524
380	2	/*ARRANGE NODES OF REDUCED ACYCLIC DIRECTED GRAPH SPECIFIED	525
381	2	BY BIT ADJACENCY MATRIX A IN ASCENDING "RANK" ORDER,	526
382	2	RESULTING IN N-ELEMENT VECTOR "ORDER" */	527

UANA 7963 US/05/77R PLIC-VLR-033078

PAGE 13

SYPT LEVEL NEST

```

383      3      DCL (AC(*),UNUSE(NN),T INIT('T'),F INIT('F')) BIT(1),
              (NORDER(NN),NODES(2) INIT(0),NEW INIT(2),OLD INIT(1))
              FIELD BIN;
384      3      DCL (DEPTH(4,NN),K INIT(0)) FIXED BIN;
385      3      DCL RANK(1,NN) FIXED BIN;
386      3      DCL (1,J,L,11,*) FIXED BIN;

/* L ---- THE NUMBER OF NODES IN THE ORIGINAL DIGRAPH */
/* NN ---- THE NUMBER OF NODES IN THE ACYCLIC DIGRAPH */
/* ORDER ---- VECTOR OF NODES IN ORIGINAL DIGRAPH IN RANK ORDER */
/* RORDER ---- VECTOR OF NODES IN ACYCLIC DIGRAPH IN RANK ORDER */
/* RANK ---- VECTOR OF RANKS IN ACYCLIC DIGRAPH (MAX DEPTH OF A
              GIVEN NODE.
              */

/* DEPTH ---- SET OF NODES IN A GIVEN RANK (OLD & NEW) IE. RANK SET */
/* NODES ---- COUNTERS FOR # OF NODES IN RANK SET (DEPTH), OLD & NEW */
/* OLD ---- POINTER TO PREVIOUS RANK SET
/* NEW ---- POINTER TO CURRENT RANK SET
/* UNUSE ---- BIT VECTOR OF NODES NOT IN CURRENT RANK SET
/* INITIALIZE RANK OF ALL NODES TO 0 */

387      3      RANK,RORDER = 0;
388      3      ORDER = 0;
/* SET UP NODES OF DEPTH 0 */
389      3      DO J=1 TO NN;
/* ENTER NODES WITHOUT PREDECESSOR IN FIRST RANK SET */
390      3      1      IF ANY(AC(*,J)) THEN GO TO OUT;
391      3      1      M,NODES(OLD) = NODES(OLD) + 1;
392      3      1      DEPTH(OLD,M)=J;
393      3      1      OUT: END;
394      3      1      IF NODES(OLD) <= J THEN GO TO ERR;
395      3      1      /* PROCEED TO FIND RANK SETS OF DEPTH 1 AND ON */
396      3      1      DO L=1 TO NN;
397      3      1      1      NODES(NEW) = 0;
398      3      1      1      UNUSE = T;
399      3      1      1      DO I=1 TO NODES(OLD);
400      3      1      1      1      II = DEPTH(OLD,I);
401      3      1      1      1      DO J=1 TO NN;
402      3      1      1      1      1      IF AC(II,J) THEN
403      3      1      1      1      1      IF UNUSE(J) THEN DO;
404      3      1      1      1      1      1      RANK(J)=L;
405      3      1      1      1      1      1      UNUSE(J)=F;
406      3      1      1      1      1      1      M,NODES(NEW)=NODES(NEW) + 1;
407      3      1      1      1      1      1      DEPTH(NEW,M)=J;
408      3      1      1      1      1      1      ENDS;
409      3      1      1      1      1      1      ENDS;
410      3      1      1      1      1      1      ENDS;
411      3      1      1      1      1      1      ENDS;
412      3      1      1      1      1      1      ENDS;
/* IF NO NODES IN NEXT RANK SET THEN REORDER: END OF RANKING */
413      3      1      1      IF NODES(NEW) <= 0 THEN GO TO REORDER;
/* EXCHANGE OLD & NEW RANK SETS */
414      3      1      1      M = NEW;
415      3      1      1      NEW = OLD;
416      3      1      1      OLD = M;
417      3      1      1

```


APPENDIX C

MODEL output corresponding to the multicountry integrated model of MINILINK presented in Chapter 7. Figure C.1 list the source descriptions augmented with generated statements by the processor (using '*' in place of the location sequence numbers). Figure C.2 shows the elements identified as members of a "compound loop" by the Cycles Analysis. Figure C.3 gives the formatted output in which the statements are grouped into corresponding subsections and the statements in each file are indented. Figure C.4 shows the cross-reference table of all names used in the MINILINK specification, and finally Figure C.5 presents the "precedence" list corresponding to this simulation example.

SOURCE LISTING AND GENERATED STATEMENTS

STATEMENT NUMBER		
1	/* SPECIFICATION OF MINILINK */	00000001
1	SEC HEADER	00000002
1	MODULE: MINILINK;	00000003
2	SOURCE: A_BANK,B_BANK,A_INPUT,B_INPUT,TRADAT;	00000004
3	TARGET: SOLUTION;	00000005
4	/*	00000006
4	REFER: A_BANK,B_BANK,A_INPUT,B_INPUT,	00000007
4	A_SOLUTION,A_A_STRUC_EQNS,B_B_STRUC_EQNS;	00000008
4	*/	00000009
4		00000010
4	SEC DATA_DESCRIPTION	00000011
4	/* DATA DESCRIPTION */	00000012
4		00000013
4	A IS MEDIA(UNIT = 3330);	00000014
5	TRADAT IS FILE(A,ORG=SEQ,FIXED);	00000015
6	SHARE IS RECORD(TRADAT);	00000016
7	ATO IS FIELD(SHARE,(NO_CNTRY,NO_CNTRY,4));	00000017
8		00000018
8	/* FROM SPECIFICATION OF COUNTRY A */	00000019
8		00000020
8	A_BANK IS FILE(A,ORG=SEQ,KEY=NUMBER,FIXED);	00000021
9	TIM_SER IS RECORD(A_BANK,(5,4));	00000022
10	NUMBER IS FIELD(TIM_SER,NUM(4));	00000023
11	LABEL IS FIELD(TIM_SER,NUM(4));	00000024
12	DATA IS FIELD(TIM_SER,DEC(14,5),(20));	00000025
13	A_INPUT IS FILE(A,ORG=SEQ,FIXED);	00000026
14	CONTROLS IS RECORD(A_INPUT,(1));	00000027
15	LINK_YR IS FIELD(CONTROLS,NUM(4));	00000028
16	LINK_NP IS FIELD(CONTROLS,NUM(4));	00000029
17	COEFF IS FIELD(CONTROLS,NUM(4),(12));	00000030
18	SOLUTION IS REPORT(A,ORG=SEQ,VARIABLE);	00000031
19	PERIOD IS GROUP(SOLUTION,(LINK_YR:END_YR));	00000032
20	HEADER IS RECORD(PERIOD,(1));	00000033
21	TIME IS FIELD(HEADER,CHAR(10));	00000034
22	NAME IS FIELD(HEADER,CHAR(20));	00000035
23	PRICE IS FIELD(HEADER,CHAR(20));	00000036
24	VALUE IS FIELD(HEADER,CHAR(20));	00000037
25	COUNTRY IS RECORD(PERIOD,(NO_CNTRY));	00000038
26	FILLER IS FIELD(COUNTRY,CHAR(10));	00000039
27	CNTRY_NAME IS FIELD(COUNTRY,CHAR(20));	00000040
28	P IS FIELD(COUNTRY,PIC*(14)Z.V(5)9');	00000041
29	V IS FIELD(COUNTRY,PIC*(14)Z.V(5)9');	00000042
30	TOTAL IS RECORD(PERIOD,(1));	00000043
31	FILLER IS FIELD(TOTAL,CHAR(50));	00000044
32	TW IS FIELD(TOTAL,PIC*(14)Z.V(5)9');	00000045
33		00000046
33	/*REDUCED COUNTRY A MODEL STRUCTURAL EQUATIONS */	00000047
33		00000048
33	A_VM(T,K)=A.COEFF(1) + A.COEFF(2) * T	00000049
33	+A.COEFF(3)*A_PM(T,K)+A.COEFF(4)*A_PM(T-1,K)	00000050
33	+A.COEFF(5)*A_X(T,K)	00000051
33	+A.COEFF(6)*A_X(T-1,K);	00000052
34	A_PX(T,K)=A.COEFF(7) + A.COEFF(8) * T	00000053
34	+A.COEFF(9)*A_PM(T,K)+A.COEFF(10)*A_PM(T-1,K)	00000054
34	+A.COEFF(11)*A_X(T,K)	00000055
34	+A.COEFF(12)*A_X(T-1,K);	00000056

Figure C.1

```

35                                     00000057
35 /* FROM COUNTRY B SPECIFICATION */ 00000058
35 B IS MEDIA(UNIT=33301);           00000059
36                                     00000060
36 B_BANK IS FILE(B,ORG=SEQ,KEY=NUMBER,FIXED); 00000061
37 TIM_SER IS RECORD(B_BANK,(5,4)); 00000062
38 B_INPUT IS FILE(B,ORG=SEQ,FIXED); 00000063
39 CONTROLS IS RECORD(B_INPUT,(1)); 00000064
40                                     00000065
40 /*REDUCED COUNTRY B MODEL STRUCTURAL EQUATIONS */ 00000066
40                                     00000067
40 B_VM(T,K)=B.COEFF(1) + B.COEFF(2) * T 00000068
40      +B.COEFF(3)*B_PM(T,K)+B.COEFF(4)*B_PM(T-1,K) 00000069
40      +B.COEFF(5)*B_X(T,K) 00000070
40      +B.COEFF(6)*B_X(T-1,K); 00000071
41 B_PX(T,K)=B.COEFF(7) + B.COEFF(8) * T 00000072
41      +B.COEFF(9)*B_PM(T,K)+B.COEFF(10)*B_PMIT-1,K) 00000073
41      +B.COEFF(11)*B_X(T,K) 00000074
41      +B.COEFF(12)*B_X(T-1,K); 00000075
42                                     00000076
42                                     00000077
42 /* END OF REFERRED DESCRIPTIONS */ 00000078
42                                     00000079
42 /* SUBSCRIPT DESCRIPTION */ 00000080
42 T IS SUBSCRIPT(PERIOD,A,LINK_YR,END_YR,1); /* SOLUTION TIME */ 00000081
43 DT IS SUBSCRIPT(DATA,1,20,1) /* DATA TIME */ 00000082
44 I IS SUBSCRIPT(A70,1,NO_CNTRY,1,1); /*COUNTRY*/ 00000083
45 J IS SUBSCRIPT(A70,1,NO_CNTRY,1,2); /*COUNTRY*/ 00000084
46 K IS SUBSCRIPT(A70,1,4,1,3); /*COMMODITY GROUP*/ 00000085
47                                     00000086
47 /* EACH COUNTRY ENDOGENOUS INTERIM VARIABLES */ 00000087
47                                     00000088
47 A_VM,B_VM,A_PX,B_PX ARE INTERIM(DEC(14,5),(LINK_YR:END_YR,4)); 00000089
51 A_X,B_X,A_PM,B_PM ARE INTERIM(DEC(14,5),(LAG_YR:END_YR,4)); 00000090
55                                     00000091
55 /* MINILINK ENDOGENOUS INTERIM VARIABLES */ 00000092
55                                     00000093
55 XL IS INTERIM(DEC(14,5),(NO_CNTRY,LAG_YR:END_YR,4)); 00000094
56 VML,PML,VXL,PXL ARE INTERIM(DEC(14,5),(NO_CNTRY,A.LINK_YR:END_YR,4)); 00000095
60 VML09 IS INTERIM(DEC(14,5),(NO_CNTRY,A.LINK_YR:END_YR)); 00000096
61                                     00000097
61 SEC ASSERTIONS 00000098
61                                     00000099
61 /*DATA REPETITION ASSERTIONS*/ 00000100
61                                     00000101
61 END_YR = A.LINK_YR + A.LINK_NP; 00000102
62 LAG_YR = A.LINK_YR - 1; 00000103
63 NO_CNTRY = 2; 00000104
64                                     00000105
64 /*ASSERTIONS FOR "CONTRACT" VARIABLES LINKING EACH COUNTRY 00000106
64 WITH MINILINK*/ 00000107
64                                     00000108
64 VML(1,T,K) = A_VM(T,K); 00000109
65 VML(2,T,K) = B_VM(T,K); 00000110
66 PXL(1,T,K) = A_PX(T,K); 00000111
67 PXL(2,T,K) = B_PX(T,K); 00000112
68                                     00000113
68 /*ASSERTIONS FOR "CONTRACT" VARIABLES LINKING MINILINK TO 00000114
68 EACH COUNTRY*/ 00000115
68                                     00000116
68 A_X(T,K) = XL(1,T,K); 00000117
69 B_X(T,K) = XL(2,T,K); 00000118
70 A_PM(T,K) = PML(1,T,K); 00000119

```


[illegible]

C.1 (Cont.)

104 \$A2002: A.SOLUTION.PERIOD.COUNTRY.FILLER="";
105 \$A2003: A.SOLUTION.PERIOD.HEADER.TIME="";
106 A IS MEDIA(UNIT=3330);
107 \$B1001 IS SUBSCRIPT((1,*1,1));
108 \$B1002 IS SUBSCRIPT((1,*1,1));
109 \$B1003 IS SUBSCRIPT((1,*1,1));
110 \$B1004 IS SUBSCRIPT((1,*1,1));
111 \$B1005 IS SUBSCRIPT((1,*1,1));
112 \$B1006 IS SUBSCRIPT((1,*1,1));

1 THE FOLLOWING GROUPS OF ITEMS ARE CIRCULARLY DESCRIBED
 (DICT#, PREFIX CODE , NAME)

0

106	SI	VXL
113	SI	VXL
62	SA	SA0020
115	SI	SI0003
183	SI	SI0003
112	SA	SA\$10003
185	SI	VML
139	SI	VML
74	SA	SA0008
140	SI	A_VM
176	SI	A_VM
91	SA	SA0001
173	SI	A_X
131	SI	A_X
70	SA	SA0012
132	SI	XL
105	SI	XL
60	SA	SA0022
107	SI	PXL
135	SI	PXL
72	SA	SA0010
136	SI	A_PX
165	SI	A_PX
80	SA	SA0002
169	SI	A_PM
127	SI	A_PM
68	SA	SA0014
128	SI	PHL
109	SI	PHL
61	SA	SA0021
111	SI	SI0002
186	SI	SI0002
108	SA	SA\$10002

Figure C.2

```

/*****
/*
/*          MINILINK MODULE DESCRIPTION
/*
/*****
MODULE: MINILINK;
SOURCE FILES: B_INPUT,B_BANK,A_INPUT,A_BANK,TRADAT;
TARGET FILES: SOLUTION;

/*****
/*
/*          DATA DESCRIPTION STATEMENTS
/*
/*****
A IS MEDIA(UNIT=3330);
B IS MEDIA(UNIT=3330);
A IS MEDIA(UNIT=3330);

/*****
/*
/*          "B_INPUT" FILE DESCRIPTION
/*
/*****
B_INPUT IS FILE(R,FIXED,SAM);
CONTROLS IS RECORD(B_INPUT,(1));
LINK_YR IS FIELD(CONTROLS,NUMERIC(4));
LINK_NP IS FIELD(CONTROLS,NUMERIC(4));
COEFF IS FIELD(CONTROLS,(12),NUMERIC(4));

/*****
/*
/*          "B_BANK" FILE DESCRIPTION
/*
/*****
B_BANK IS FILE(P,KEY=NUMBER,FIXED,SAM);
TIM_SER IS RECORD(B_BANK,(5,4));
NUMBER IS FIELD(TIM_SER,NUMERIC(4));
LABEL IS FIELD(TIM_SER,NUMERIC(4));
DATA IS FIELD(TIM_SER,(20),DECIMAL(14,5));

/*****
/*
/*          "SOLUTION" FILE DESCRIPTION
/*
/*****
SOLUTION IS FILE(A,VARIABLE,SAM);
PERIOD IS GROUP(SOLUTION,(LINK_YR:END_YR));
HEADER IS RECORD(PERIOD,(1));
TIME IS FIELD(HEADER,CHAR(10));
NAME IS FIELD(HEADER,CHAR(20));
PRICE IS FIELD(HEADER,CHAR(20));
VALUE IS FIELD(HEADER,CHAR(20));
COUNTRY IS RECORD(PERIOD,(NO_CNTRY));
FILLER IS FIELD(COUNTRY,CHAR(10));
DATA_DESCRIPTION IS FIELD(COUNTRY,CHAR(20));
P IS FIELD(COUNTRY,PICTURE '(14)Z.V(5)9');
V IS FIELD(COUNTRY,PICTURE '(14)Z.V(5)9');

```

```

00000001
00000002
00000003
00000004
00000005
00000006
00000007
00000008
00000009
00000010
00000011
00000012
00000013
00000014
00000015
00000016
00000017
00000100
00000200
00000300
00000301
00000302
00000303
00000304
00000305
00000306
00000307
00000400
00000500
00000600
00000700
00000800
00000801
00000802
00000803
00000804
00000805
00000806
00000807
00000900
00001000
00001100
00001200
00001300
00001301
00001302
00001303
00001304
00001305
00001306
00001307
00001400
00001500
00001600
00001700
00001800
00001900
00002000
00002100
00002200
00002300
00002400
00002500

```

Figure C.3


```

END_YR IS FIELD;                                00004200
$10003 IS INTERIM;                             00004300
$10002 IS INTERIM;                             00004400
$10001 IS INTERIM;                             00004500
LAG_YR IS INTERIM;                             00004600
NO_CNTRY IS FIELD;                             00004700
VML09 IS INTERIM(*,NO_CNTRY,A,LINK_YR:END_YR),DECIMAL(14,5)); 00004800
VML IS INTERIM(END_CNTRY,A,LINK_YR:END_YR,4),DECIMAL(14,5)); 00004900
PML IS INTERIM(END_CNTRY,A,LINK_YR:END_YR,4),DECIMAL(14,5)); 00005000
VXL IS INTERIM(END_CNTRY,A,LINK_YR:END_YR,4),DECIMAL(14,5)); 00005100
PXL IS INTERIM(END_CNTRY,A,LINK_YR:END_YR,4),DECIMAL(14,5)); 00005200
XL IS INTERIM(NO_CNTRY,LAG_YR:END_YR,4),DECIMAL(14,5)); 00005300
A_X IS INTERIM(LAG_YR:END_YR,4),DECIMAL(14,5)); 00005400
R_X IS INTERIM(LAG_YR:END_YR,4),DECIMAL(14,5)); 00005500
A_PM IS INTERIM(LAG_YR:END_YR,4),DECIMAL(14,5)); 00005600
R_PM IS INTERIM(LAG_YR:END_YR,4),DECIMAL(14,5)); 00005700
A_VM IS INTERIM(LINK_YR:END_YR,4),DECIMAL(14,5)); 00005800
R_VM IS INTERIM(LINK_YR:END_YR,4),DECIMAL(14,5)); 00005900
A_PX IS INTERIM(LINK_YR:END_YR,4),DECIMAL(14,5)); 00006000

```

```

      B_PX IS INTERIM((LINK_YR:END_YR,4),DECIMAL(14,5));
00006100
00006101
00006102
00006103
00006104
00006105
00006106
00006107
00006200
00006300
00006400
00006500
00006600
00006700
00006800
00006900
00007000
00007100
00007101
00007102
00007103
00007104
00007105
00007106
00007107
00007108
00007109
00007110
00007111
00007112
00007113
00007114
00007115
00007116
00007117
00007118
00007119
00007120
00007121
00007122
00007123
00007124
00007125
00007126
00007127
00007128
00007129
00007130
00007131
00007132
00007133
00007134
00007135
00007136
00007137
00007138
00007139
00007140
00007141
00007142
00007143
00007144
00007145

/******
/*
/*          SUBSCRIPT DESCRIPTION STATEMENTS
/*
/******
FOREACH$HEADER IS SUBSCRIPT(HEADER,(1,1,1,1));
FOREACH$COUNTRY IS SUBSCRIPT(COUNTRY,(1,NO_CNTRY,1,1));
FOREACH$TOTAL IS SUBSCRIPT(TOTAL,(1,1,1,1));
FOREACH$CONTROLS IS SUBSCRIPT(CONTROLS,(1,1,1,1));
FOREACH$PERIOD IS SUBSCRIPT(PERIOD,(LINK_YR,END_YR,1,1));
FOREACH$VML09 IS SUBSCRIPT(VML09,(1,*,1,1));
K IS SUBSCRIPT(VXL,(1,4,1,3));
J IS SUBSCRIPT(A70,(1,NO_CNTRY,1,2));
I IS SUBSCRIPT(VXL,(1,NO_CNTRY,1,1));
T IS SUBSCRIPT(VXL,(A.LINK_YR,END_YR,1,2));

/******
/*
/*          ASSERTIONS
/*
/******

$A0001:
  $I001=PML(I,T,K) * VML(J,T,K) / VML09(I,T,K);

$A0028:
  VML09(FOREACH$VML09,I,T)=SUM(VML(I,T,K),K);

$A0017:
  B_PH(LAG_YR,K)=B_BANK.DATA(I,K,LAG_YR);

$A0018:
  A_PH(LAG_YR,K)=A_BANK.DATA(I,K,LAG_YR);

$A0017:
  B_X(LAG_YR,K)=B_BANK.DATA(5,K,LAG_YR);

$A0016:
  A_X(LAG_YR,K)=A_BANK.DATA(5,K,LAG_YR);

$A0015:
  B_PH(T,K)=PML(2,T,K);

$A0013:
  B_X(T,K)=XL(2,T,K);

$A0011:
  PXL(2,T,K)=B_PH(T,K);

$A0009:
  VML(2,T,K)=B_VM(T,K);

$A0005:
  LAG_YR=A_INPUT.LINK_YR(FOREACH$CONTROLS) - 1;

$A0004:
  B_PX(T,K)=B_INPUT.COEFF(K,7) + B_INPUT.COEFF(K,8) * T +
    B_INPUT.COEFF(K,9) * B_PH(T,K) + B_INPUT.COEFF(K,10) * B_PH(T-
    1,K) + B_INPUT.COEFF(K,11) * B_X(T,K) + B_INPUT.COEFF(K,12) *

```

C.3 (Cont.)

```

      B_X(T-1,K);                                00007146
$A0003:                                           00007147
      B_V4(T,K)=B_INPUT.COEFF(K,1) + B_INPUT.COEFF(K,2) * T + 00007148
      B_INPUT.COEFF(K,3) * B_PM(T,K) + B_INPUT.COEFF(K,4) * B_PM(T-1,00007150
      K) + B_INPUT.COEFF(K,5) * B_X(T,K) + B_INPUT.COEFF(K,6) * B_X(00007151
      T-1,K);                                     00007152
$A0023:                                           00007153
      PERIOD(T)=T;                               00007154
$A0024:                                           00007155
      NAME(FOR EACH$PERIOD,T)='NAME';            00007156
$A0025:                                           00007157
      PRICE(FOR EACH$PERIOD,T)='PRICE IMPORT';    00007158
$A0026:                                           00007159
      VALUE(FOR EACH$PERIOD,T)='VALUE EXPORT';    00007160
$A0027:                                           00007161
      DATA_DESCRIPTION(T,I)=I;                  00007162
$A0029:                                           00007163
      P(T,I)=SUM($I0001);                         00007164
$A0030:                                           00007165
      V(T,I)=SUM(VXL(I,T,K),K);                   00007166
$A0031:                                           00007167
      TW(I,T)=SUM(V(I,T),I);                      00007168
$A0007:                                           00007169
      NO_CNTRY=2;                                 00007170
$A0005:                                           00007171
      END_YR=A_INPUT.LINK_YR(FOR EACH$CONTROLS) + A_INPUT.LINK_NP( 00007172
      F$REACH$CONTROLS);                          00007173
/*****                                           00007174
/*                                           00007175
/* SIMULTANEOUS ASSERTIONS                      /* 00007176
/*                                           /* 00007177
/*****                                           00007178
$A0020:                                           00007179
      VXL(I,T,K)=SUM($I0003,J);                   00007180
$A$I0003:                                           00007181
      $I0003=A70(I,J,K) * V4L(J,T,K);             00007182
$A0008:                                           00007183
      V4L(I,T,K)=A_V4(T,K);                       00007184
$A0001:                                           00007185
      A_V4(T,K)=A_INPUT.COEFF(FOR EACH$CONTROLS,1) + A_INPUT.COEFF( 00007186
      F$REACH$CONTROLS,2) * T + A_INPUT.COEFF(F$REACH$CONTROLS,3) * 00007187
      A_PM(T,K) + A_INPUT.COEFF(FOR EACH$CONTROLS,4) * A_PM(T-1,K) + 00007188
      A_INPUT.COEFF(FOR EACH$CONTROLS,5) * A_X(T,K) + A_INPUT.COEFF( 00007189
      F$REACH$CONTROLS,6) * A_X(T-1,K);            00007190
$A0012:                                           00007191

```

C.3 (Cont.)

Figure C.4

```

A_X(T,K)=XL(I,T,K);
$A0022:
XL(I,T,K)=VXL(I,T,K) / PXL(I,T,K);
$A0010:
PXL(I,T,K)=A_PX(T,K);
$A0002:
A_PX(T,K)=A_INPUT.COEFF(FOR EACH$CONTROLS,7) + A_INPUT.COEFF(
  F$EACH$CONTROLS,9) * T + A_INPUT.COEFF(FOR EACH$CONTROLS,9) *
  A_PM(T,K) + A_INPUT.COEFF(FOR EACH$CONTROLS,10) * A_PA(T-1,K) +
  A_INPUT.COEFF(FOR EACH$CONTROLS,11) * A_X(T,K) + A_INPUT.COEFF(
  F$EACH$CONTROLS,12) * A_X(T-1,K);
$A0014:
A_PM(T,K)=PML(I,T,K);
$A0021:
PML(J,T,K)=SUM($I0002,I);
$A$I0002:
$I0002=PXL(I,T,K) * A70(I,J,K);
/*****/
/*
/*          END OF SIMULTANEDUS GROUP
/*
/*
/*****/

```

00007209
00007210
00007211
00007212
00007213
00007214
00007215
00007216
00007217
00007218
00007219
00007220
00007221
00007222
00007223
00007224
00007225
00007226
00007227
00007228
00007229
00007230
00007231
00007232
00007233
*/ 00007234
*/ 00007235
*/ 00007236
*/ 00007237
00007238

ATTRIBUTES AND CROSS-REFERENCE TABLE
ATTRIBUTE AND REFERENCES

SYMT NO.	ACL NO.	DICT NO.	IDENTIFIER	
96	000000	92	\$A\$10001	ASSERTION NAME
98	000000	108	\$A\$10002	ASSERTION NAME
100	000000	112	\$A\$10003	ASSERTION NAME
33	52	91	\$A0001	ASSERTION NAME
34	56	90	\$A0002	ASSERTION NAME
40	71	79	\$A0003	ASSERTION NAME
41	75	78	\$A0004	ASSERTION NAME
61	102	77	\$A0005	ASSERTION NAME
62	103	76	\$A0006	ASSERTION NAME
63	104	75	\$A0007	ASSERTION NAME
64	109	74	\$A0008	ASSERTION NAME
65	110	73	\$A0009	ASSERTION NAME
66	111	72	\$A0010	ASSERTION NAME
67	112	71	\$A0011	ASSERTION NAME
69	117	70	\$A0012	ASSERTION NAME
69	119	69	\$A0013	ASSERTION NAME
70	119	68	\$A0014	ASSERTION NAME
71	120	67	\$A0015	ASSERTION NAME
72	125	66	\$A0016	ASSERTION NAME
73	126	65	\$A0017	ASSERTION NAME
74	127	64	\$A0018	ASSERTION NAME
75	128	63	\$A0019	ASSERTION NAME
76	132	62	\$A0020	ASSERTION NAME
77	133	61	\$A0021	ASSERTION NAME
78	134	60	\$A0022	ASSERTION NAME
79	137	59	\$A0023	ASSERTION NAME
80	138	58	\$A0024	ASSERTION NAME
81	139	57	\$A0025	ASSERTION NAME
82	140	56	\$A0026	ASSERTION NAME

Figure C.4

83	141	55	\$A0027	ASSERTION NAME	
84	142	54	\$A0028	ASSERTION NAME	
85	143	53	\$A0029	ASSERTION NAME	
86	144	52	\$A0030	ASSERTION NAME	
87	145	51	\$A0031	ASSERTION NAME	
107	00000	200	FOREACHSVNLO9	(1,*,1,1) IN VNLO9(=NO_CNTRY,A,LINK_YREND_YR), SUBSCRIPT	84
108	00000	201	FOREACHSPERIOD	(LINK_YR,END_YR,1,1) IN PERIOD(LINK_YREND_YR) IN SOLUTION IN A, SUBSCRIPT	80,81,82
109	00000	202	FOREACHSCONROLS	(1,1,1,1) IN CONTROLS(1) IN A_INPUT IN A, SUBSCRIPT	33,34,40,41,61,62
110	00000	203	FOREACHSTOTAL	(1,1,1,1) IN TOTAL(1) IN PERIOD(LINK_YREND_YR) IN SOLUTION IN A, SUBSCRIPT	103
111	00000	204	FOREACHSCOUNTRY	(1,NO_CNTRY,1,1) IN COUNTRY(=CNTRY) IN PERIOD(LINK_YREND_YR) IN SOLUTION	104
112	00000	205	FOREACHSHEADER	(1,1,1,1) IN HEAD(1) IN PERIOD(LINK_YREND_YR) IN SOLUTION IN A, SUBSCRIPT	105
97	00000	94	\$10001	INTERIM	85,96,97
99	00000	110	\$10002	INTERIM	77,98,99
101	00000	114	\$10003	INTERIM	76,100,101
104	00000	199	A	MEDIA	18
4	14	4	A	MEDIA	5,8,13,42,56,57,58,59,60
8	21	9	A_BANK	IN A, FILE	9
13	26	8	A_INPUT	IN A, FILE	14
52	90	45	A_PM	(LAG_YREND_YR,4) INTERIM(DECIMAL(14,5))	33,33,34,34,52,52,70,74
48	89	49	A_PX	(LINK_YREND_YR,4) INTERIM(DECIMAL(14,5))	34,48,66
50	89	47	A_VM	(LINK_YREND_YR,4) INTERIM(DECIMAL(14,5))	33,50,64
54	90	43	A_X	(LAG_YREND_YR,4) INTERIM(DECIMAL(14,5))	33,33,34,34,54,54,68,72

C.4 (Cont.)

[illegible]

20	33	16	HEADER	(1) IN PERIOD(LINK_YREND_YR) IN SOLUTION IN A, RECORD 21,22,23,24
44	83	99	J	(1,NO_CTRY,1,1) IN VAL(NO_CTRY,A_LINK_YREND_YR,4), SUBSCRIPT 83
45	84	191	J	(1,NO_CTRY,1,2) IN AT0(NO_CTRY,NO_CTRY,4) IN SHARE IN TRADAT IN A, SUBSCRIPT
46	85	190	K	(1,4,1,3) IN VAL(NO_CTRY,A_LINK_YREND_YR,4), SUBSCRIPT
94	86	86	LABEL	IN TIM_SER(5,4) IN B_BANK IN B, FIELD(NUmeric(4))
11	24	34	LABEL	IN TIM_SER(5,4) IN A_BANK IN A, FIELD(NUmeric(4))
89	88	2	LAG_YR	INTERIM 51,51,52,52,53,53,54,54,55,55,55,72,73,74,75,89 51,52,62
91	83	83	LINK_MP	IN CONTROLS(1) IN B_INPUT IN B, FIELD(NUmeric(4))
16	29	31	LINK_MP	IN CONTROLS(1) IN A_INPUT IN A, FIELD(NUmeric(4)) 16,61
90	82	82	LINK_YR	IN CONTROLS(1) IN B_INPUT IN B, FIELD(NUmeric(4)) 19,47,48,49,50
15	28	32	LINK_YR	IN CONTROLS(1) IN A_INPUT IN A, FIELD(NUmeric(4)) 15,19,19,47,47,48,49,49,50,50,61,62
1	3	0	MINILINK	MODULE NAME
22	25	28	NAME	IN HEADER(1) IN PERIOD(LINK_YREND_YR) IN SOLUTION IN A, FIELD(CHAR(20)) 22,80
88	81	1	NO_CTRY	INTERIM 7,25,44,45,45,55,55,56,56,57,57,58,58,59,59,60,60,88 7,25,44,63
93	85	85	NUMBER	IN TIM_SER(5,4) IN B_BANK IN B, FIELD(NUmeric(4))
10	23	35	NUMBER	IN TIM_SER(5,4) IN A_BANK IN A, FIELD(NUmeric(4))
28	41	23	P	IN CCOUNTY(NO_CTRY) IN PERIOD(LINK_YREND_YR) IN SOLUTION IN A, FIELD PICTURE '(11A12.V15)9', 28,85
19	32	14	PERIOD	(LINK_YREND_YR) IN SOLUTION IN A, GROUP 19,20,25,30,79
58	95	39	P4L	(NO_CTRY,A_LINK_YREND_YR,4) INTERIM(DECIMAL(14,5)) 58,70,71,77,96
23	36	27	PRICE	IN HEADER(1) IN PERIOD(LINK_YREND_YR) IN SOLUTION IN A, FIELD(CHAR(20)) 23,81
56	95	41	P4L	(NO_CTRY,A_LINK_YREND_YR,4) INTERIM(DECIMAL(14,5)) 56,56,66,67,78,98
6	16	19	SHARE	IN TRADAT IN A, RECORD 7

C.4 (Cont.)

28	31	7	SOLUTION	IN A, FILE
42	81	104	T	19
37	62	12	TIM_SER	(A, LINK_YR, END_YR, 1, 2) IN VALIND_CNTRY, A, LINK_YR, END_YR, 4), SUBSCRIPT 33, 34, 40, 41, 79
9	22	18	TIM_SER	(5, 4) IN B, BANK IN B, RECORD 93, 94, 95
21	34	29	TIME	(5, 4) IN A, BANK IN A, RECORD 10, 11, 12
30	43	13	TOTAL	IN HEADER(1) IN PERIOD(LINK_YR, END_YR) IN SOLUTION IN A, FIELD(CHAR(10)) 21, 105
5	15	10	TRADAT	(1) IN PERIOD(LINK_YR, END_YR) IN SOLUTION IN A, RECORD 31, 32
32	45	20	TW	IN A, FILE 6
29	42	22	V	IN TOTAL(1) IN PERIOD(LINK_YR, END_YR) IN SOLUTION IN A, FIELD(PICTURE '(14 2-V(5)39') 32, 87
24	37	26	VALUE	IN CNTRY(IND_CNTRY) IN PERIOD(LINK_YR, END_YR) IN SOLUTION IN A, FIELD PICTURE '(14)2-V(5)9') 29, 80, 87
59	95	38	VXL	IN HEADER(1) IN PERIOD(LINK_YR, END_YR) IN SOLUTION IN A, FIELD(CHAR(20)) 24, 82
60	96	37	VXL09	IND_CNTRY, A, LINK_YR, END_YR, 4) INTERIMDECIMAL(14, 5)) 56, 57, 60, 65, 84, 86, 100
57	95	40	VXL	(A, IND_CNTRY, A, LINK_YR, END_YR) INTERIMDECIMAL(14, 5)) 60, 84, 96
55	94	42	KL	IND_CNTRY, A, LINK_YR, END_YR, 4) INTERIMDECIMAL(14, 5)) 57, 76, 78, 86
				IND_CNTRY, LAG_YR, END_YR, 4) INTERIMDECIMAL(14, 5)) 55, 66, 69, 78

C.4 (Cont.)

PRECEDENCE MATRIX		PREDECESSORS	
DICTO	SUCCESSORS		
1	37(6) 38(6)	39(6) 40(6) 41(6) 42(6) 191(6) 99(6) 141(2)	
2	42(6) 43(6)	44(6) 45(6) 46(6)	142(2)
3	51(1) 61(1)		
4	37(6) 38(6)	39(6) 40(6) 41(6) 104(6) 81(1) 91(1)	
5	11(1) 12(1)		
6	12(1) 5(9)		3(1) 6(9)
7	199(2)		3(1)
8	17(1)		14(2)
9	18(1)		4(1)
10	19(1)		9(9)
11	21(1)		10(9)
12	22(1)		4(1)
13	23(1)		5(1)
14	7(2)		6(1)
15	14(2)		20(2)
16	14(2) 15(9)		13(2)
17	30(1) 31(1)		15(2) 16(2) 32(6) 143(6) 103(2)
18	32(1)		11(6) 22(2) 23(2) 24(2) 25(2) 141(6) 16(9)
19	34(1) 35(1)		26(2) 27(2) 28(2) 29(2)
20	13(2)		8(1)
21	13(2) 20(9)		10(1)
22	15(2)		11(1)
23	15(2) 22(9)		12(1)
24	15(2) 23(9)		13(1)
25	15(2) 24(9)		14(1)
26	15(2)		15(1)
27	15(2) 26(9)		16(1)
28	15(2) 27(9)		17(1)
29	15(2) 28(9)		18(1)
30	16(1) 167(1)		19(1)
31	179(1) 180(1) 181(1) 182(1)	183(1) 174(1) 177(1) 178(1)	20(1)
32	183(1) 30(9)	49(6) 50(6) 143(1) 31(9)	21(1)
33	184(1) 124(1)		22(1)
34	185(1)		23(1)
35	186(1)		24(1)
36	187(1)		25(1)
37	188(1)		26(1)
38	189(1)		27(1)
39	190(1)		28(1)
40	191(1)		29(1)
41	192(1)		30(1)
42	193(1)		31(1)
43	194(1)		32(1)
44	195(1)		33(1)
45	196(1)		34(1)
46	197(1)		35(1)
47	198(1)		36(1)
48	199(1)		37(1)
49	200(1)		38(1)
50	201(1)		39(1)
51	202(1)		40(1)
52	203(1)		41(1)
53	204(1)		42(1)
54	205(1)		43(1)
55	206(1)		44(1)
56	207(1)		45(1)
57	208(1)		46(1)
58	209(1)		47(1)
59	210(1)		48(1)
60	211(1)		49(1)
61	212(1)		50(1)
62	213(1)		51(1)
63	214(1)		52(1)
64	215(1)		53(1)
65	216(1)		54(1)
66	217(1)		55(1)
67	218(1)		56(1)
68	219(1)		57(1)
69	220(1)		58(1)
70	221(1)		59(1)
71	222(1)		60(1)
72	223(1)		61(1)
73	224(1)		62(1)
74	225(1)		63(1)
75	226(1)		64(1)
76	227(1)		65(1)
77	228(1)		66(1)
78	229(1)		67(1)
79	230(1)		68(1)
80	231(1)		69(1)
81	232(1)		70(1)
82	233(1)		71(1)
83	234(1)		72(1)
84	235(1)		73(1)
85	236(1)		74(1)
86	237(1)		75(1)
87	238(1)		76(1)
88	239(1)		77(1)
89	240(1)		78(1)
90	241(1)		79(1)
91	242(1)		80(1)
92	243(1)		81(1)
93	244(1)		82(1)
94	245(1)		83(1)
95	246(1)		84(1)
96	247(1)		85(1)
97	248(1)		86(1)
98	249(1)		87(1)
99	250(1)		88(1)
100	251(1)		89(1)
101	252(1)		90(1)
102	253(1)		91(1)
103	254(1)		92(1)
104	255(1)		93(1)
105	256(1)		94(1)
106	257(1)		95(1)
107	258(1)		96(1)
108	259(1)		97(1)
109	260(1)		98(1)
110	261(1)		99(1)
111	262(1)		100(1)
112	263(1)		101(1)
113	264(1)		102(1)
114	265(1)		103(1)
115	266(1)		104(1)
116	267(1)		105(1)
117	268(1)		106(1)
118	269(1)		107(1)
119	270(1)		108(1)
120	271(1)		109(1)
121	272(1)		110(1)
122	273(1)		111(1)
123	274(1)		112(1)
124	275(1)		113(1)
125	276(1)		114(1)
126	277(1)		115(1)
127	278(1)		116(1)
128	279(1)		117(1)
129	280(1)		118(1)
130	281(1)		119(1)
131	282(1)		120(1)
132	283(1)		121(1)
133	284(1)		122(1)
134	285(1)		123(1)
135	286(1)		124(1)
136	287(1)		125(1)
137	288(1)		126(1)
138	289(1)		127(1)
139	290(1)		128(1)
140	291(1)		129(1)
141	292(1)		130(1)
142	293(1)		131(1)
143	294(1)		132(1)
144	295(1)		133(1)
145	296(1)		134(1)
146	297(1)		135(1)
147	298(1)		136(1)
148	299(1)		137(1)
149	300(1)		138(1)
150	301(1)		139(1)
151	302(1)		140(1)
152	303(1)		141(1)
153	304(1)		142(1)
154	305(1)		143(1)
155	306(1)		144(1)
156	307(1)		145(1)
157	308(1)		146(1)
158	309(1)		147(1)
159	310(1)		148(1)
160	311(1)		149(1)
161	312(1)		150(1)
162	313(1)		151(1)
163	314(1)		152(1)
164	315(1)		153(1)
165	316(1)		154(1)
166	317(1)		155(1)
167	318(1)		156(1)
168	319(1)		157(1)
169	320(1)		158(1)
170	321(1)		159(1)
171	322(1)		160(1)
172	323(1)		161(1)
173	324(1)		162(1)
174	325(1)		163(1)
175	326(1)		164(1)
176	327(1)		165(1)
177	328(1)		166(1)
178	329(1)		167(1)
179	330(1)		168(1)
180	331(1)		169(1)
181	332(1)		170(1)
182	333(1)		171(1)
183	334(1)		172(1)
184	335(1)		173(1)
185	336(1)		174(1)
186	337(1)		175(1)
187	338(1)		176(1)
188	339(1)		177(1)
189	340(1)		178(1)
190	341(1)		179(1)
191	342(1)		180(1)
192	343(1)		181(1)
193	344(1)		182(1)
194	345(1)		183(1)
195	346(1)		184(1)
196	347(1)		185(1)
197	348(1)		186(1)
198	349(1)		187(1)
199	350(1)		188(1)
200	351(1)		189(1)
201	352(1)		190(1)
202	353(1)		191(1)
203	354(1)		192(1)
204	355(1)		193(1)
205	356(1)		194(1)
206	357(1)		195(1)
207	358(1)		196(1)
208	359(1)		197(1)
209	360(1)		198(1)
210	361(1)		199(1)
211	362(1)		200(1)
212	363(1)		201(1)
213	364(1)		202(1)
214	365(1)		203(1)
215	366(1)		204(1)
216	367(1)		205(1)
217	368(1)		206(1)
218	369(1)		207(1)
219	370(1)		208(1)
220	371(1)		209(1)
221	372(1)		210(1)
222	373(1)		211(1)
223	374(1)		212(1)
224	375(1)		213(1)
225	376(1)		214(1)
226	377(1)		215(1)
227	378(1)		216(1)
228	379(1)		217(1)
229	380(1)		218(1)
230	381(1)		219(1)
231	382(1)		220(1)
232	383(1)		221(1)
233	384(1)		222(1)
234	385(1)		223(1)
235	386(1)		224(1)
236	387(1)		225(1)
237	388(1)		226(1)
238	389(1)		227(1)
239	390(1)		228(1)
240	391(1)		229(1)
241	392(1)		230(1)
242	393(1)		231(1)
243	394(1)		232(1)
244	395(1)		233(1)
245	396(1)		234(1)
246	397(1)		235(1)
247	398(1)		236(1)
248	399(1)		237(1)
249	400(1)		238(1)
250	401(1)		239(1)
251	402(1)		240(1)
252	403(1)		241(1)
253	404(1)		242(1)
254	405(1)		243(1)
255	406(1)		244(1)
256	407(1)		245(1)
257	408(1)		246(1)
258	409(1)		247(1)
259	410(1)		248(1)
260	411(1)		249(1)
261	412(1)		250(1)
262	413(1)		251(1)
263	414(1)		252(1)
264	415(1)		253(1)
265	416(1)		254(1)
266	417(1)		255(1)
267	418(1)		256(1)
268	419(1)		257(1)
269	420(1)		258(1)
270	421(1)		259(1)
271	422(1)		260(1)
272	423(1)		261(1)
273	424(1)		262(1)
274	425(1)		263(1)
275	426(1)		264(1)
276	427(1)		265(1)
277	428(1)		266(1)
278	429(1)		267(1)
279	430(1)		268(1)
280	431(1)		269(1)
281	432(1)		270(1)
282	433(1)		271(1)
283	434(1)		272(1)
284	435(1)		273(1)
285	436(1)		274(1)
286	437(1)		275(1)
287	438(1)		276(1)
288	439(1)		277(1)
289	440(1)		278(1)
290	441(1)		279(1)
291	442(1)		280(1)
292	443(1)		281(1)
293	444(1)		282(1)
294	445(1)		283(1)
295	446(1)		284(1)
296	447(1)		285(1)
297	448(1)		286(1)
298	449(1)		287(1)
299	450(1)		288(1)
300	451(1)		289(1)
301	452(1)		290(1)
302	453(1)		291(1)
303	454(1)		292(1)
304	455(1)		293(1)
305	456(1)		294(1)
306	457(1)		295(1)
307	458(1)		296(1)
308	459(1)		297(1)
309	460(1)		298(1)
310	461(1)		299(1)
311	462(1)		300(1)
312	463(1)		301(1)
313	464(1)		302(1)
314	465(1)		303(1)
315	466(1)		304(1)
316	467(1)		305(1)
317	468(1)		306(1)
318	469(1)		307(1)
319	470(1)		308(1)
320	47		

50	81(4)	32(6)	82(6)	143(6)	147(2)
51	90(4)	97(3)			
52	93(4)	95(3)			
53	98(4)	97(3)	200(3)		
54	100(4)	94(3)			
55	101(4)	201(3)			
56	102(4)	201(3)			
57	103(4)	201(3)			
58	104(4)	201(3)			
59	105(4)	201(3)			
60	106(4)	106(3)	107(3)		
61	107(4)	111(3)			
62	108(4)	111(3)			
63	109(4)	117(3)	118(3)		
64	110(4)	117(3)	120(3)		
65	111(4)	117(3)	122(3)		
66	112(4)	117(3)	124(3)		
67	113(4)	126(3)			
68	114(4)	126(3)			
69	115(4)	133(3)			
70	116(4)	133(3)			
71	117(4)	133(3)			
72	118(4)	133(3)			
73	119(4)	133(3)			
74	120(4)	133(3)			
75	121(4)	133(3)			
76	122(4)	133(3)			
77	123(4)	133(3)			
78	124(4)	133(3)			
79	125(4)	145(3)	202(3)	202(3)	150(3)
80	126(4)	145(3)	149(3)	151(3)	153(3)
81	127(4)	155(3)	150(3)	152(3)	154(3)
82	128(4)	155(3)	151(3)	153(3)	154(3)
83	129(4)	155(3)	152(3)	154(3)	155(3)
84	130(4)	155(3)	153(3)	155(3)	156(3)
85	131(4)	155(3)	154(3)	156(3)	157(3)
86	132(4)	155(3)	155(3)	157(3)	158(3)
87	133(4)	155(3)	156(3)	158(3)	159(3)
88	134(4)	155(3)	157(3)	159(3)	160(3)
89	135(4)	155(3)	158(3)	160(3)	161(3)
90	136(4)	155(3)	159(3)	161(3)	162(3)
91	137(4)	155(3)	160(3)	162(3)	163(3)
92	138(4)	155(3)	161(3)	163(3)	164(3)
93	139(4)	155(3)	162(3)	164(3)	165(3)
94	140(4)	155(3)	163(3)	165(3)	166(3)
95	141(4)	155(3)	164(3)	166(3)	167(3)
96	142(4)	155(3)	165(3)	167(3)	168(3)
97	143(4)	155(3)	166(3)	168(3)	169(3)
98	144(4)	155(3)	167(3)	169(3)	170(3)
99	145(4)	155(3)	168(3)	170(3)	171(3)
100	146(4)	155(3)	169(3)	171(3)	172(3)
101	147(4)	155(3)	170(3)	172(3)	173(3)
102	148(4)	155(3)	171(3)	173(3)	174(3)
103	149(4)	155(3)	172(3)	174(3)	175(3)
104	150(4)	155(3)	173(3)	175(3)	176(3)
105	151(4)	155(3)	174(3)	176(3)	177(3)
106	152(4)	155(3)	175(3)	177(3)	178(3)
107	153(4)	155(3)	176(3)	178(3)	179(3)
108	154(4)	155(3)	177(3)	179(3)	180(3)
109	155(4)	155(3)	178(3)	180(3)	181(3)
110	156(4)	155(3)	179(3)	181(3)	182(3)
111	157(4)	155(3)	180(3)	182(3)	183(3)
112	158(4)	155(3)	181(3)	183(3)	184(3)
113	159(4)	155(3)	182(3)	184(3)	185(3)
114	160(4)	155(3)	183(3)	185(3)	186(3)
115	161(4)	155(3)	184(3)	186(3)	187(3)
116	162(4)	155(3)	185(3)	187(3)	188(3)
117	163(4)	155(3)	186(3)	188(3)	189(3)
118	164(4)	155(3)	187(3)	189(3)	190(3)
119	165(4)	155(3)	188(3)	190(3)	191(3)
120	166(4)	155(3)	189(3)	191(3)	192(3)
121	167(4)	155(3)	190(3)	192(3)	193(3)
122	168(4)	155(3)	191(3)	193(3)	194(3)
123	169(4)	155(3)	192(3)	194(3)	195(3)
124	170(4)	155(3)	193(3)	195(3)	196(3)
125	171(4)	155(3)	194(3)	196(3)	197(3)
126	172(4)	155(3)	195(3)	197(3)	198(3)
127	173(4)	155(3)	196(3)	198(3)	199(3)
128	174(4)	155(3)	197(3)	199(3)	200(3)
129	175(4)	155(3)	198(3)	200(3)	201(3)
130	176(4)	155(3)	199(3)	201(3)	202(3)
131	177(4)	155(3)	200(3)	202(3)	203(3)
132	178(4)	155(3)	201(3)	203(3)	204(3)
133	179(4)	155(3)	202(3)	204(3)	205(3)
134	180(4)	155(3)	203(3)	205(3)	206(3)
135	181(4)	155(3)	204(3)	206(3)	207(3)
136	182(4)	155(3)	205(3)	207(3)	208(3)
137	183(4)	155(3)	206(3)	208(3)	209(3)
138	184(4)	155(3)	207(3)	209(3)	210(3)
139	185(4)	155(3)	208(3)	210(3)	211(3)
140	186(4)	155(3)	209(3)	211(3)	212(3)
141	187(4)	155(3)	210(3)	212(3)	213(3)
142	188(4)	155(3)	211(3)	213(3)	214(3)
143	189(4)	155(3)	212(3)	214(3)	215(3)
144	190(4)	155(3)	213(3)	215(3)	216(3)
145	191(4)	155(3)	214(3)	216(3)	217(3)
146	192(4)	155(3)	215(3)	217(3)	218(3)
147	193(4)	155(3)	216(3)	218(3)	219(3)
148	194(4)	155(3)	217(3)	219(3)	220(3)
149	195(4)	155(3)	218(3)	220(3)	221(3)
150	196(4)	155(3)	219(3)	221(3)	222(3)
151	197(4)	155(3)	220(3)	222(3)	223(3)
152	198(4)	155(3)	221(3)	223(3)	224(3)
153	199(4)	155(3)	222(3)	224(3)	225(3)
154	200(4)	155(3)	223(3)	225(3)	226(3)
155	201(4)	155(3)	224(3)	226(3)	227(3)
156	202(4)	155(3)	225(3)	227(3)	228(3)
157	203(4)	155(3)	226(3)	228(3)	229(3)
158	204(4)	155(3)	227(3)	229(3)	230(3)
159	205(4)	155(3)	228(3)	230(3)	231(3)
160	206(4)	155(3)	229(3)	231(3)	232(3)
161	207(4)	155(3)	230(3)	232(3)	233(3)
162	208(4)	155(3)	231(3)	233(3)	234(3)
163	209(4)	155(3)	232(3)	234(3)	235(3)
164	210(4)	155(3)	233(3)	235(3)	236(3)
165	211(4)	155(3)	234(3)	236(3)	237(3)
166	212(4)	155(3)	235(3)	237(3)	238(3)
167	213(4)	155(3)	236(3)	238(3)	239(3)
168	214(4)	155(3)	237(3)	239(3)	240(3)
169	215(4)	155(3)	238(3)	240(3)	241(3)
170	216(4)	155(3)	239(3)	241(3)	242(3)
171	217(4)	155(3)	240(3)	242(3)	243(3)
172	218(4)	155(3)	241(3)	243(3)	244(3)
173	219(4)	155(3)	242(3)	244(3)	245(3)
174	220(4)	155(3)	243(3)	245(3)	246(3)
175	221(4)	155(3)	244(3)	246(3)	247(3)
176	222(4)	155(3)	245(3)	247(3)	248(3)
177	223(4)	155(3)	246(3)	248(3)	249(3)
178	224(4)	155(3)	247(3)	249(3)	250(3)
179	225(4)	155(3)	248(3)	250(3)	251(3)
180	226(4)	155(3)	249(3)	251(3)	252(3)
181	227(4)	155(3)	250(3)	252(3)	253(3)
182	228(4)	155(3)	251(3)	253(3)	254(3)
183	229(4)	155(3)	252(3)	254(3)	255(3)
184	230(4)	155(3)	253(3)	255(3)	256(3)
185	231(4)	155(3)	254(3)	256(3)	257(3)
186	232(4)	155(3)	255(3)	257(3)	258(3)
187	233(4)	155(3)	256(3)	258(3)	259(3)
188	234(4)	155(3)	257(3)	259(3)	260(3)
189	235(4)	155(3)	258(3)	260(3)	261(3)
190	236(4)	155(3)	259(3)	261(3)	262(3)
191	237(4)	155(3)	260(3)	262(3)	263(3)
192	238(4)	155(3)	261(3)	263(3)	264(3)
193	239(4)	155(3)	262(3)	264(3)	265(3)
194	240(4)	155(3)	263(3)	265(3)	266(3)
195	241(4)	155(3)	264(3)	266(3)	267(3)
196	242(4)	155(3)	265(3)	267(3)	268(3)
197	243(4)	155(3)	266(3)	268(3)	269(3)
198	244(4)	155(3)	267(3)	269(3)	270(3)
199	245(4)	155(3)	268(3)	270(3)	271(3)
200	246(4)	155(3)	269(3)	271(3)	272(3)

105	42(2)	132(8)			60(4)
106	62(3)				113(8)
107	62(3)	106(3)			135(8)
108	186(6)				107(3)
109	39(2)	128(8)	188(8)		184(3)
110					61(4)
111	61(3)				186(2)
112	151(4)				186(8)
113	40(2)	91(8)	106(8)		184(3)
114					62(4)
115	62(3)				183(2)
116	44(2)	151(8)	153(8)		183(8)
117	61(3)	64(3)	65(3)	66(3)	63(4)
118	61(3)				142(8)
119	45(2)	168(8)	171(8)		87(1)
120	64(3)				84(4)
121	44(2)	155(8)	157(8)		31(1)
122	65(3)				62(4)
123	43(2)	173(8)	175(8)		87(1)
124	66(3)				66(4)
125	44(2)	151(8)	153(8)		34(1)
126	67(3)				67(4)
127	45(2)	169(8)			68(4)
128	68(3)				105(8)
129	44(2)	155(8)	157(8)		69(4)
130	67(3)				
131	43(2)	173(8)			70(4)
132	71(3)				105(8)
133	41(2)				71(4)
134	71(3)				147(8)
135	41(2)	107(4)			72(4)
136	72(3)				165(8)
137	39(2)				73(4)
138	71(3)				156(8)
139	35(2)	97(8)	185(8)		74(4)
140	74(3)				176(8)
141	37(6)	38(6)	39(6)	40(6)	75(4)
142	15(6)	36(6)	-1(2)	41(6)	
143	42(6)	43(6)	44(6)	45(6)	76(4)
144	75(3)	77(3)	47(6)	48(6)	121(1)
145	104(6)				151(2)
146	104(6)	144(2)			77(4)
147	52(2)	134(8)			31(1)
148	74(3)				78(4)
149	74(3)				84(1)
150	74(3)				84(1)
151	74(3)	79(3)			116(8)
152	74(3)				84(1)
153	75(3)	79(3)			116(8)
154	74(3)				84(1)
155	74(3)	79(3)			121(8)
156	74(3)				84(1)
157	74(3)	79(3)			121(8)
158	44(2)	134(8)			79(4)
159	77(3)				84(1)
160	79(3)				84(1)
161	79(3)				84(1)
162	79(3)				84(1)

C.5 (Cont.)

163	79(3)		84(1)
164	79(3)		84(1)
165	80(2)	136(8)	80(4)
166	80(3)		30(1)
167	80(3)		30(1)
168	82(3)	81(3)	30(1)
169	82(3)		117(8)
170	82(3)	81(3)	30(1)
171	83(3)	81(3)	119(8)
172	83(3)	81(3)	30(1)
173	83(3)	81(3)	123(8)
174	83(3)	81(3)	30(1)
175	83(3)	81(3)	123(8)
176	82(2)	140(8)	81(4)
177	81(3)		30(1)
178	81(3)		30(1)
179	81(3)		30(1)
180	81(3)		30(1)
181	81(3)		30(1)
182	81(3)		30(1)
183	115(2)	115(4)	122(4)
184	115(3)	112(3)	30(1)
185	115(3)	112(3)	135(8)
186	115(2)	111(4)	108(4)
187	94(2)	95(8)	92(4)
188	92(3)		109(8)
189	92(3)		56(8)
190			116)
191			141(6)
192			
193	175(4)		203(3)
194	21(2)		195(4)
195	176(4)		204(3)
196	25(2)		195(4)
197	194(4)		205(3)
198	29(2)		197(4)
199	54(3)		7(2)
200	54(3)		
201	54(3)	57(3)	
202	75(3)	77(3)	
203	193(3)	58(3)	
204	195(3)	78(3)	
205	197(3)	79(3)	
206	197(3)	80(3)	
207	197(3)	81(3)	

APPENDIX D

This Appendix list the PL/1 source code of the matrix and vector manipulation functions, random variates generators and other model building utility procedures as described in chapter 8.

WANA 3514 04/14/78 PLIC-VLR-033078
VERSION 5.5

05/360 PL/I COMPILER (F) ON VROS

PAGE
DATE 78.

STMT	LEVEL	NEST			
1			MMVERTD: PROCEDURE (R,A,N,ZERO);		00010000
			/*		00020000
			PROCEDURE OBTAINS INVERSE OF SQUARE SYMMETRIC MATRIX		00030000
			OF MOMENTS AND COMPUTES COEFFICIENTS U = INV(A)*X*Y		00040000
			(IE TO BE USED IN OLS LIKE CALCULATIONS)		00050000
			A(N,N) - INPUT MATRIX OF MOMENTS.		00060000
			R(N,N) - OUTPUT MATRIX WITH INVERSE AND COEFFICIENTS		00070000
			IN R(1:N-1,N).		00080000
			*/		00090000
2	1		DCL (R(*,*),A(*,*),ZERO,DET,RIK,RR) BIN FLOAT(53);		00100000
3	1		DCL (K,I,J,N) BIN FIXED;		00110000
4	1		DO I=1 TO N;		00120000
5	1	1	DO J=1 TO N;		00130000
6	1	2	R(I,J) = A(I,J);		00140000
7	1	2	END;		00150000
8	1	1	END;		00160000
9	1		DET = 1.;		00170000
10	1		DO K=1 TO N-1;		00180000
11	1	1	DET = DET*A(K,K);		00190000
			/*		00200000
			----- MESSAGES REFERRING TO "MODEL" STATEMENT NUMBER		00210000
			CAN BE INTRODUCED BY CODE GENERATION AS FOLLOWS:		00220000
			DICT# - DICTIONARY NUMBER OF CURRENT STATEMENT		00230000
			CALLING FUNCTION.		00240000
			%INCLUDE INCL16 (USEASS);		00250000
			DCL ACADR ENTRY (BIN FIXED,POINTER);		00260000
			CALL ACADR (DICT#,STORAGE_PTR)		00270000
					00280000
			USE%LINE# &PLIN# CAN THEN BE PASSED AS PARAMETERS		00290000
			TO PROCEDURE.		00300000
			*/		00310000
12	1	1	IF DET < ZERO THEN RETURN;		00320000
13	1	1	RR = 1./DET;		00330000
14	1	1	R(K,K)=1.;		00340000
15	1	1	DO J=1 TO N;		00350000
16	1	1	R(K,J) = RR*A(K,J);		00360000
17	1	2	END;		00370000
18	1	2	DO I=1 TO N-1;		00380000
19	1	1	IF K=I THEN GO TO CONT;		00390000
20	1	2	RIK = R(I,K);		00400000
21	1	2	R(I,K)=0.;		00410000
22	1	2	DO J=1 TO N;		00420000
23	1	3	R(I,J) = R(I,J) - RIK*R(K,J);		00430000
24	1	3	END;		00440000
25	1	2	CONT: END;		00450000
26	1	1	END;		00460000
27	1		END MMVERTD;		00470000

ATTRIBUTE TABLE

WANA 351R J4/14/78 PLIC-VER-033078
VERSION 5.5

OS/360 PL/I COMPILER (F) ON VMOS

PAGE
DATE 78.1

STMT	LEVEL	NEST			
1			MOMENTD: PROCEDURE (R,A,M,N);		00010000
			/*		00020000
			COMPUTES ALL MOMENTS OF MATRIX A(M,N): R = A'A		00030000
			*/		00040000
2	1		DCL (A(*,*),K(*,*)) BIN FLOAT(53),		00050000
			(M,N,I,J,K) BIN FIXED,		00060000
			S BIN FLOAT(53);		00070000
3	1		DO I=1 TO N;		00080000
4	1	1	DO J=1 TO N;		00090000
5	1	2	S=0.;		00100000
6	1	2	DO K=1 TO M;		00110000
7	1	3	S = S + A(K,I) * A(K,J);		00120000
8	1	3	END;		00130000
9	1	2	R(I,J),R(J,I) = S;		00140000
10	1	2	END;		00150000
11	1	1	END;		00160000
12	1		RETURN;		00170000
13	1		END MOMENTD;		00180000

DCL	IDENTIFIER	ATTRIBUTE TABLE	ATTRIBUTES
2	A	(*,*)	PARAMETER ALIGNED BIN FLOAT(LONG)
2	I	AUTO	ALIGNED BIN FIXED(15,0)
2	J	AUTO	ALIGNED BIN FIXED(15,0)
2	K	AUTO	ALIGNED BIN FIXED(15,0)
2	M	PARAMETER	ALIGNED BIN FIXED(15,0)
1	MOMENTD	ENTRY	BIN FIXED(15,0)
2	N	PARAMETER	ALIGNED BIN FIXED(15,0)
2	R	(*,*)	PARAMETER ALIGNED BIN FLOAT(LONG)
2	S	AUTO	ALIGNED BIN FLOAT(LONG)

STORAGE REQUIREMENTS.

THE STORAGE AREA (IN STATIC) FOR THE PROCEDURE LABELLED MOMENTD IS 244 BYTES LONG.
THE PROGRAM CSECT IS NAMED MOMENTD AND IS 566 BYTES LONG.
THE STATIC CSECT IS NAMED MOMENTDA AND IS 332 BYTES LONG.

*STATISTICS= SOURCE RECORDS = 18, PROG TEXT STMTS = 13, OBJECT BYTES = 586

COMPILER DIAGNOSTICS.

WARNINGS.

IEH05201 1 OPTION MAIN HAS NOT BEEN SPECIFIED FOR THE EXTERNAL PROCEDURE, STATEMENT NUMBER 1.

GANA 3613 04/14/78 PLIC-VER-033078
VERSION 5.5

05/360 PL/I COMPILER (F) ON VMOS

PAGE
DATE 7c

STMT LEVEL NEST

1			INVD: PROCEDURE (R,A,N,ZERO);	00010000
			/*	00020000
			MATRIX INVERSION PROCEDURE FOR GENERAL SQUARE MATRIX	00030000
			USING MAXIMUM PIVOT STRATEGY.	00040000
			A(N,N) - INPUT MATRIX.	00050000
			R(N,N) - RETURNED INVERSE OF A.	00060000
			ZERO - ZERO DEFINITION FOR TOLERANCE.	00070000
			*/	00080000
2	1		DCL (R(*,*),A(*,*),ZERO,DET,SUM,RMS,TEST,X,PIV,PIV1,TOL,TEMP)	00090000
			BIN FLOAT(53);	00100000
3	1		DCL (I,J,IR(N),IC(N),KP,S,V,L,N,M,IRANK) BIN FIXED;	00110000
			/*	00120000
			----- MESSAGES REFERRING TO "MODEL" STATEMENT NUMBER	00130000
			CAN BE INTRODUCED BY CODE GENERATION AS FOLLOWS:	00140000
			DICT# - DICTIONARY NUMBER OF CURRENT STATEMENT	00150000
			CALLING FUNCTION.	00160000
			INCLUDE INCLIN (USEASS);	00170000
			DCL ACROD ENTRY (BIN FIXED, POINTER);	00180000
			CALL ACRD (DICT#, STORAGE_PTR)	00190000
				00200000
			USER_LINE# & PLIN# CAN THEN BE PASSED AS PARAMETERS	00210000
			TO PROCEDURE.	00220000
			*/	00230000
4	1		DET = 1.;	00240000
5	1		SUM = 0.;	00250000
6	1		DO I=1 TO N;	00260000
7	1	1	DO J=1 TO N;	00270000
8	1	2	R(I,J)=A(I,J);	00280000
9	1	2	SUM=SUM+R(I,J)**2;	00290000
10	1	2	END;	00300000
11	1	1	END;	00310000
12	1		SUM=SQRT(SUM);	00320000
13	1		RMS=SUM/4**2;	00330000
14	1		TOL=ZERO+RMS;	00340000
15	1		S=0.;	00350000
16	1		IR,IC=0.;	00360000
17	1		RR=N;	00370000
18	1		DO WHILE (S<KP);	00380000
19	1	1	I,J=0;	00390000
20	1	1	TEST=0.;	00400000
21	1	1	DO K=1 TO N;	00410000
22	1	2	IF IR(K) **2 THEN GO TO OUTR;	00420000
23	1	2	DO L=1 TO N;	00430000
24	1	3	IF IC(L) **2 THEN GO TO OUTC;	00440000
25	1	3	X=ABS(R(K,L));	00450000
26	1	3	IF X < TEST THEN GO TO OUTC;	00460000
27	1	3	I=K;	00470000
28	1	3	J=L;	00480000
29	1	3	TEST=X;	00490000
30	1	3	OUTC: END;	00500000
31	1	2	OUTR: END;	00510000
32	1	1		
33	1			
34	1			

GANA 3613 04/14/78 PLIC-VER-033078

PAGE

STMT LEVEL NEST

35	1	1	PIV=R(I,J);	00520000
36	1	1	DET=PIV*DET;	00530000
37	1	1	IF (ABS(PIV) <= TOL) THEN GO TO SING;	00540000
39	1	1	IK(I)=J;	00550000
40	1	1	IC(J)=I;	00560000
41	1	1	PIV=1./PIV;	00570000
42	1	1	R(I,J)=PIV;	00580000
43	1	1	DO K=1 TO N;	00590000
44	1	2	IF K=J THEN R(I,K)=R(I,K)+PIV;	00600000
46	1	2	END;	00610000
47	1	1	DO K=1 TO N;	00620000
48	1	2	IF K=I THEN GO TO OUT;	00630000
50	1	2	PIV1=R(K,J);	00640000
51	1	2	DO L=1 TO N;	00650000
52	1	3	IF L=J THEN R(K,L)=R(K,L) - PIV1*R(I,L);	00660000
54	1	3	END;	00670000
55	1	2	OUT: END;	00680000
56	1	1	DO K=1 TO N;	00690000
57	1	2	IF K=I THEN R(K,J)=-PIV*R(K,J);	00700000
59	1	2	END;	00710000
60	1	1	S = S + 1;	00720000
61	1	1	END;	00730000
62	1	1	DO I=1 TO N;	00740000
63	1	2	K=IC(I);	00750000
64	1	2	M=IR(I);	00760000
65	1	2	IF K=I THEN GO TO LAST;	00770000
67	1	2	DET=-DET;	00780000
68	1	2	DO L=1 TO N;	00790000
69	1	3	TEMP=R(K,L);	00800000
70	1	3	R(K,L)=R(I,L);	00810000
71	1	3	R(I,L)=TEMP;	00820000
72	1	3	END;	00830000
73	1	2	DO L=1 TO N;	00840000
74	1	3	TEMP=R(L,M);	00850000
75	1	3	R(L,M)=R(L,I);	00860000
76	1	3	R(L,I)=TEMP;	00870000
77	1	3	END;	00880000
78	1	2	IC(M)=K;	00890000
79	1	2	IR(K)=M;	00900000
80	1	1	LAST: END;	00910000
81	1	1	SING: IANK=S;	00920000
82	1	1	END MINVD;	00930000

ATTRIBUTE TABLE

DCL IDENTIFIER

2 A
2 ABS
2 DET

ATTRIBUTES

(*,*) PARAMETER ALIGNED BIN FLOAT(LONG)
GENERIC BUILT-IN FUNCTION
AUTO ALIGNED BIN FLOAT(LONG)

GANA 5525 04/17/78 PLIC-VER-033078
VERSION 5.5

OS/360 PL/I COMPILER (F) ON VMOS

PAGE
DATE 78.

STMT LEVEL NEST

```

1          EIGEND: PROCEDURE (R,A,N,MFREQ,MMAX,VZERO,EPS);
/*
   POWER METHOD TO DETERMINE EIGENVALUES AND EIGENVECTORS
   OF AN N*N REAL MATRIX A.
   A(N,N) - MATRIX WHOSE EIGENVALUES ARE REQUIRED.
   B(N,N) - REPEATED PRODUCT MATRIX, B=(A-LAMDA(1)*I)
             (A-LAMDA(2)*I),...
   C,D - AUXILIARY MATRICES.
   EPS - TOLERANCE USED IN CONVERGENCE TEST.
   IDENT(N,N) - IDENTITY MATRIX.
   REORTHOGONALIZATIONS OF V.
   MMAX - MAXIMUM NUMBER OF ITERATIONS.
   V - CURRENT APPROXIMATION TO EIGENVECTOR.
   VZERO - STARTING GUESS FOR EIGENVECTOR V.
   Y - AUXILIARY VECTOR.
   R(N+1,N) - OUTPUT MATRIX WITH EIGENVECTORS IN R(2:N+1,*)
             AND EIGENVALUES IN R(1,*).
   ----- COULD BE MODIFIED TO USE STRUCTURE
             DCL 1 R,
                 2 LAMDA(N),*EIGENVALUES*
                 2 U(N,N) BIN FLOAT(53);*EIGENVECTORS*
   ----- FOR MORE CONVENIENCE
   M - ITERATION COUNTER.
   MFREQ - NUMBER OF ITERATIONS BETWEEN PERIODIC
*/
2          1      DCL (R(*,*),A(*,*),VZERO(*),EPS,B(N,N),IDENT(N,N),V(N),Y(N),
3                  LZERO,L,C(N,N),D(N,N)) 9IN FLOAT(53);
4                  DCL (N,MFREQ,MMAX,1,J,K,IM1) 6IN FIXED;
5                  DCL MATVECD ENTRY;
6                  DCL VECLEND ENTRY;
7                  DCL SCAVECD ENTRY;
8                  DCL SCAMATO ENTRY;
9                  DCL MATSUBD ENTRY;
10                 DCL MATMLTD ENTRY;
11                 IDENT,B,R=0.;
12                 DO I=1 TO N;
13                     IDENT(I,I)=1.;
14                 END;
/* --- SET U EQ TO IDENTITY MATRIX --- */
15                 CALL MATVECD(B,IDENT,N,N);
/* --- POWER METHOD FOR ALL EIGENVALUES --- */
16                 DO J=1 TO N;
/* --- MODIFY STARTING VECTOR SO THAT IT IS ORTHOGONAL TO ALL
                     PREVIOUSLY COMPUTED EIGENVALUES --- */
17                     CALL MATVECD(V,VZERO,N,N);
18                     CALL VECLEND(LZERO,V,N);
/* --- PERFORM SUCCESSIVE POWER METHOD ITERATIONS --- */
19                     DO M=1 TO MMAX;
/* --- PERIODICALLY RE-ORTHOGONALIZE THE VECTOR V --- */
20                     K=MOD(N,MFREQ);

```

GANA 5525 04/17/78 PLIC-VER-033078

PAGE

STMT LEVEL NEST

```

21 1 2 IF K=0 THEN DO;
23 1 3 CALL MATVECD(Y,B,V,N,N);
24 1 3 CALL VECLEND(L,Y,N);
25 1 3 CALL SCAVECD(V,1.0/L,Y,N);
26 1 3 END;
/* --- COMPUTE NEW VECTOR V AND ITS LENGTH --- */
27 1 2 CALL MATVECD(Y,A,V,N,N);
28 1 2 CALL VECLEND(L,Y,N);
29 1 2 CALL SCAVECD(V,1.0/L,Y,N);
/* --- CONVERGENCE TEST --- */
30 1 2 IF ABS((L-LZERO)/LZERO) < EPS THEN GO TO CONVERG;
32 1 2 LZERO=L;
33 1 2 END;
/* --- SAVE PARTIAL RESULTS IF METHOD DID NOT CONVERGE --- */
34 1 1 IM1=I-1;
/*
----- MESSAGES REFERRING TO "MODEL" STATEMENT NUMBER
CAN BE INTRODUCED BY CODE GENERATION AS FOLLOWS:
DICT# - DICTIONARY NUMBER OF CURRENT STATEMENT
CALLING FUNCTION.
XINCLUDE INCLIB (DSEASS);
DCL ACROD ENTRY (BIN FIXED, POINTER);
CALL ACROD (DICT#, STORAGE_PTR)

USER_LIN# & PLIN# CAN THEN BE PASSED AS PARAMETERS
TO PROCEDURE.
*/
35 1 1 PUT SKIP LIST ("NO CONVERGENCE. PARTIAL RESULTS ONLY");
36 1 1 PUT SKIP EDIT ("VALIDITY RANK (1-1) IS:",IM1)
(A,F(5));
37 1 1 RETURN;
/* --- ESTABLISH SIGN OF EIGENVALUE --- */
38 1 1 CONVERG: CALL MATVECD(Y,A,V,N,N);
39 1 1 DO K=1 TO N;
40 1 2 IF (ABS(V(K)) < 1.0E-3) THEN GO TO ZERO;
42 1 2 IF (V(K)*Y(K) < 0.0) THEN L=-L;
44 1 2 GO TO STORE;
45 1 2 ZERO: END;
/* --- STORE CURRENT EIGENVALUE & EIGENVECTOR --- */
46 1 1 STORE: R(1,1)=L;
47 1 1 DO K=2 TO N+1;
48 1 2 R(K,1)=V(K-1);
49 1 2 END;
/* --- MODIFY MATRIX B --- */
50 1 1 IF I<N THEN DO;
52 1 2 CALL SCAMATD(C,L,IDENT,N,N);
53 1 2 CALL MATSUBD(C,A,C,N,N);
54 1 2 CALL MATLTD(C,D,B,N,N,N);
55 1 2 CALL MATEWD(B,C,N,N);
56 1 2 END;
57 1 1 END;

```

00520000
00530000
00540000
00550000
00560000
00570000
00580000
00590000
00600000
00610000
00620000
00630000
00640000
00650000
00660000
00670000
00680000
00690000
00700000
00710000
00720000
00730000
00740000
00750000
00760000
00770000
00780000
00790000
00800000
00810000
00820000
00830000
00840000
00850000
00860000
00870000
00880000
00890000
00900000
00910000
00920000
00930000
00940000
00950000
00960000
00970000
00980000
00990000
01000000
01010000
01020000

GANA 5525 04/17/78 PLIC-VER-033078

PAGE

STMT LEVEL NEST

SB 1 END EIGEND;

01030000

ATTRIBUTE TABLE	
DCL	IDENTIFIER
2	A
	ABS
2	B
2	C
38	CONVERG
2	D
1	EIGEND
2	EPS
3	I
2	IDENT
3	IMI
3	J
3	K
2	L
2	LZERO
3	M
4	MATEWD
10	MATMLTD
9	MATSUBD
5	MATVECD
3	MFREQ
3	MMAX
	MOD
3	N
2	R
8	SCANATD
7	SCAVECD
46	STORE
	SYSPRINT
2	V
6	VECLEND
2	VZERO
2	Y
45	ZERO
ATTRIBUTES	
(*,*) PARAMETER ALIGNED BIN FLOAT(LONG)	
GENERIC BUILT-IN FUNCTION	
(*,*) AUTO ALIGNED BIN FLOAT(LONG)	
(*,*) AUTO ALIGNED BIN FLOAT(LONG)	
STMT LABEL CONSTANT	
(*,*) AUTO ALIGNED BIN FLOAT(LONG)	
ENTRY DEC FLOAT(SHORT)	
PARAMETER ALIGNED BIN FLOAT(LONG)	
AUTO ALIGNED BIN FIXED(15,0)	
(*,*) AUTO ALIGNED BIN FLOAT(LONG)	
AUTO ALIGNED BIN FIXED(15,0)	
AUTO ALIGNED BIN FIXED(15,0)	
AUTO ALIGNED BIN FIXED(15,0)	
AUTO ALIGNED BIN FLOAT(LONG)	
AUTO ALIGNED BIN FLOAT(LONG)	
AUTO ALIGNED BIN FIXED(15,0)	
EXT ENTRY BIN FIXED(15,0)	
EXT ENTRY BIN FIXED(15,0)	
EXT ENTRY BIN FIXED(15,0)	
EXT ENTRY BIN FIXED(15,0)	
PARAMETER ALIGNED BIN FIXED(15,0)	
PARAMETER ALIGNED BIN FIXED(15,0)	
GENERIC BUILT-IN FUNCTION	
PARAMETER ALIGNED BIN FIXED(15,0)	
(*,*) PARAMETER ALIGNED BIN FLOAT(LONG)	
EXT ENTRY DEC FLOAT(SHORT)	
EXT ENTRY DEC FLOAT(SHORT)	
STMT LABEL CONSTANT	
FILE EXT	
(*) AUTO ALIGNED BIN FLOAT(LONG)	
EXT ENTRY DEC FLOAT(SHORT)	
(*) PARAMETER ALIGNED BIN FLOAT(LONG)	
(*) AUTO ALIGNED BIN FLOAT(LONG)	
STMT LABEL CONSTANT	

AGGREGATE LENGTH TABLE		
STATEMENT NO.	IDENTIFIER	LENGTH IN BYTES
2	M	ADJUSTABLE
2	C	ADJUSTABLE
2	D	ADJUSTABLE
2	IDENT	ADJUSTABLE

GANA 3499 04/14/76 PLIC-VER-033078
VERSION 5.5

05/360 PL/I COMPILER (F) ON VMS

PAGE
DATE 70.1

STMT	LEVEL	NEST		
1			UNIFORM: PROCEDURE (A,B);	00010000
			/*	00020000
			GENERATES UNIFORMLY DISTRIBUTED VARIATES IN THE RANGE	00030000
			BETWEEN A AND B.	00040000
			*/	00050000
2	1		DCL (A,B,X) FLOAT(16);	00060000
			/*	00070000
			RANNOL - INSTALLATION PROVIDED UNIF(0,1) RANDOM	00080000
			NUMBER GENERATOR.	00090000
			*/	00100000
3	1		DCL RANNOL RETURNS (FLOAT(16));	00110000
4	1		X = A + (B-A)*RANNOL;	00120000
5	1		RETURN (X);	00130000
6	1		END UNIFORM;	00140000
				00150000
				00160000

ATTRIBUTE TABLE

DCL	IDENTIFIER	ATTRIBUTES
2	A	PARAMETER ALIGNED DEC FLOAT(LONG)
2	B	PARAMETER ALIGNED DEC FLOAT(LONG)
3	RANNOL	EXT ENTRY DEC FLOAT(LONG)
1	UNIFORM	ENTRY DEC FLOAT(SHORT)
2	X	AUTO ALIGNED DEC FLOAT(LONG)

STORAGE REQUIREMENTS.

THE STORAGE AREA (IN STATIC) FOR THE PROCEDURE LABELLED UNIFORM IS 216 BYTES LONG.
THE PROGRAM CSECT IS NAMED UNIFORM AND IS 314 BYTES LONG.
THE STATIC CSECT IS NAMED UNIFORMA AND IS 304 BYTES LONG.

STATISTICS SOURCE RECORDS = 16, PROG TEXT STMTS = 6, OBJECT BYTES = 314

COMPILER DIAGNOSTICS.

WARNINGS.

IEW05261 1 OPTION MAIN HAS NOT BEEN SPECIFIED FOR THE EXTERNAL PROCEDURE, STATEMENT NUMBER 1.

END OF DIAGNOSTICS.

ELAPSED TIME .01 MINS

GAMA 3500 04/14/78 PLIC-VER-033078
VERSION 5.5

OS/360 PL/1 COMPILER (F) ON VMOS

PAGE
DATE 78.1

STMT LEVEL NEST

1			GAMMA: PROCEDURE (K,A);	00010000
			/*	00020000
			GENERATES GAMMA DISTRIBUTED VARIATES WITH PARAMETERS	00030000
			"K" AND "A".	00040000
			*/	00050000
2	1		DCL (TR,X,A) FLOAT(16);	00060000
3	1		DCL (K,I) BIN FIXED;	00070000
			/*	00080000
			RANNOL - INSTALLATION PROVIDED UNIF(0,1) RANDOM	00090000
			NUMBER GENERATOR.	00100000
			*/	00110000
4	1		DCL RANNOL RETURNS (FLOAT(16));	00120000
5	1		TR = 1.0;	00130000
6	1		DO J=1 TO K;	00140000
7	1	1	TR = TR*RANNOL;	00150000
8	1	1	END;	00160000
9	1		X = -LOG(TR)/A;	00170000
10	1		RETURN (X);	00180000
11	1		END GAMMA;	00190000
				00200000
				00210000

ATTRIBUTE TABLE

DCL IDENTIFIER

ATTRIBUTES

2	A	PARAMETER ALIGNED DEC FLOAT(LONG)
1	GAMMA	ENTRY DEC FLOAT(SHORT)
3	I	AUTO ALIGNED BIN FIXED(15,0)
3	K	PARAMETER ALIGNED BIN FIXED(15,0)
	LOG	GENERIC BUILT-IN FUNCTION
4	RANNOL	EXT ENTRY DEC FLOAT(LONG)
2	TR	AUTO ALIGNED DEC FLOAT(LONG)
2	X	AUTO ALIGNED DEC FLOAT(LONG)

STORAGE REQUIREMENTS.

THE STORAGE AREA (IN STATIC) FOR THE PROCEDURE LABELLED GAMMA IS 228 BYTES LONG.
THE PROGRAM CSECT IS NAMED GAMMA AND IS 413 BYTES LONG.
THE STATIC CSECT IS NAMED *-GAMMAA AND IS 324 BYTES LONG.

STATISTICS SOURCE RECORDS = 21, PROG TEXT STMTS = 11, OBJECT BYTES = 418

COMPILER DIAGNOSTICS.

WARNINGS.

GANA 3501 04/14/78 PLIC-VER-033078
VERSION 5.5

05/360 PL/I COMPILER (F) ON VMOS

PAGE
DATE 78.10

STMT LEVEL NEST

1		NORMAL: PROCEDURE (EX,STD _X);	00010000
		/*	00020000
		GENERATES NORMALLY DISTRIBUTED VARIATES WITH MEAN "EX"	00030000
		AND STANDARD DEVIATION "STD _X ".	00040000
		*/	00050000
2	1	DCL (EX,STD _X ,S,X) FLOAT(16);	00060000
3	1	DCL I BIN FIXED;	00070000
		/*	00080000
		RANNOL - INSTALLATION PROVIDED UNIF(0,1) RANDOM	00090000
		NUMBER GENERATOR.	00100000
		*/	00110000
4	1	DCL RANNOL RETURNS (FLOAT(16));	00120000
5	1	S = 0.0;	00130000
6	1	DO I=1 TO 12;	00140000
7	1	S = S + RANNOL;	00150000
8	1	END;	00160000
9	1	X = STD _X *(S - 6.0) + EX;	00170000
10	1	RETURN (X);	00180000
11	1	END NORMAL;	00190000
			00200000
			00210000

ATTRIBUTE TABLE

DCL	IDENTIFIER	ATTRIBUTES
2	EX	PARAMETER ALIGNED DEC FLOAT(LONG)
3	I	AUTO ALIGNED BIN FIXED(15,0)
1	NORMAL	ENTRY BIN FIXED(15,0)
4	RANNOL	EXT ENTRY DEC FLOAT(LONG)
2	S	AUTO ALIGNED DEC FLOAT(LONG)
2	STD _X	PARAMETER ALIGNED DEC FLOAT(LONG)
2	X	AUTO ALIGNED DEC FLOAT(LONG)

STORAGE REQUIREMENTS.

THE STORAGE AREA (IN STATIC) FOR THE PROCEDURE LABELLED NORMAL IS 228 BYTES LONG.
THE PROGRAM CSECT IS NAMED NORMAL AND IS 413 BYTES LONG.
THE STATIC CSECT IS NAMED *NORMALA AND IS 348 BYTES LONG.

STATISTICS SOURCE RECORDS = 21, PROG TEXT STMTS = 11, OBJECT BYTES = 418

COMPILER DIAGNOSTICS.

WARNINGS.

IEWJ5261 1 OPTION MAIN HAS NOT BEEN SPECIFIED FOR THE EXTERNAL PROCEDURE, STATEMENT NUMBER 1.

GAN 3494 04/14/78 PLIC-VER-033078
VERSION 5.5

OS/360 PL/I COMPILER (F) ON VMOS

PAGE
DATE 70

STMT	LEVEL	NEST		
1			EXPON: PROCEDURE (EX);	00010000
			/*	00020000
			GENERATES EXPONENTIALLY DISTRIBUTED VARIATES WITH	00030000
			MEAN EX.	00040000
			*/	00050000
2	1		DCL (EX,R,X) FLOAT(16);	00060000
			/*	00070000
			RANNOL - INSTALLATION PROVIDED UNIF(0,1) RANDOM	00080000
			NUMBER GENERATOR.	00090000
			*/	00100000
3	1		DCL RANNOL RETURNS (FLOAT(16));	00110000
4	1		P = RANNOL;	00120000
5	1		X = -EX * LOG(R);	00130000
6	1		RETURN (X);	00140000
7	1		END EXPON;	00150000
				00160000
				00170000

ATTRIBUTE TABLE

DCL	IDENTIFIER	ATTRIBUTES
2	EX	PARAMETER ALIGNED DEC FLOAT(LONG)
1	EXPON	ENTRY DEC FLOAT(SHORT)
	LOG	GENERIC BUILT-IN FUNCTION
2	R	AUTO ALIGNED DEC FLOAT(LONG)
3	RANNOL	EXT ENTRY DEC FLOAT(LONG)
2	X	AUTO ALIGNED DEC FLOAT(LONG)

STORAGE REQUIREMENTS.

THE STORAGE AREA (IN STATIC) FOR THE PROCEDURE LABELLED EXPON IS 224 BYTES LONG.
THE PROGRAM CSECT IS NAMED EXPON AND IS 338 BYTES LONG.
THE STATIC CSECT IS NAMED **EXPONA AND IS 312 BYTES LONG.

STATISTICS SOURCE RECORDS = 17, PPOG TEXT STMTS = 7, OBJECT BYTES = 338

COMPILER DIAGNOSTICS.

WARNINGS.

IFM05261 1 OPTION MAIN HAS NOT BEEN SPECIFIED FOR THE EXTERNAL PROCEDURE, STATEMENT NUMBER 1.

END OF DIAGNOSTICS.

ELAPSED TIME .32 MINS

GANA 3505 04/14/78 PLIC-VER-033078
VERSION 5.5

OS/360 PL/I COMPILER (F) ON VROS

PAGE
DATE 78.11

STMT	LEVEL	NEST		
1			MATMLTD: PROCEDURE (R,A,B,M,N,L);	00010000
2	1		/* --- R(M,L) = A(M,N) * B(N,L) --- */	00020000
3	1		DCL (K(*,*),A(*,*),B(*,*),S) BIN FLOAT(53);	00030000
4	1		DCL (I,J,K,M,N,L) BIN FIXED;	00040000
5	1	1	DO I=1 TO M;	00050000
6	1	2	DO J=1 TO L;	00060000
7	1	2	R(I,J) = 0.;	00070000
8	1	1	END;	00080000
9	1	1	DO I=1 TO M;	00090000
10	1	1	DO J=1 TO L;	00100000
11	1	2	DO K=1 TO N;	00110000
12	1	3	R(I,J)=A(I,K)*B(K,J)+R(I,J);	00120000
13	1	2	END;	00130000
14	1	1	END;	00140000
15	1	1	END;	00150000
16	1		RETURN;	00160000
17	1		MATADD: ENTRY (R,A,B,M,N);	00170000
18	1		/* --- R(M,N) = A(M,N) * B(N,N) --- */	00180000
19	1	1	DO I=1 TO M;	00190000
20	1	2	DO J=1 TO N;	00200000
21	1	2	R(I,J) = A(I,J)*B(I,J);	00210000
22	1	1	END;	00220000
23	1	1	END;	00230000
24	1		RETURN;	00240000
25	1		MATSUED: ENTRY (R,A,B,M,N);	00250000
26	1		/* --- R(M,N) = A(M,N) - B(N,N) --- */	00260000
27	1	1	DO I=1 TO M;	00270000
28	1	2	DO J=1 TO N;	00280000
29	1	2	R(I,J) = A(I,J) - B(I,J);	00290000
30	1	1	END;	00300000
31	1	1	END;	00310000
32	1		RETURN;	00320000
33	1		MATEQD: ENTRY (R,A,M,N);	00330000
34	1		/* --- R(M,N) = A(M,N) --- */	00340000
35	1	1	DO I=1 TO M;	00350000
36	1	2	DO J=1 TO N;	00360000
37	1	2	R(I,J) = A(I,J);	00370000
38	1	1	END;	00380000
39	1	1	END;	00390000
40	1		RETURN;	00400000
41	1		TRANSPD: ENTRY (R,A,M,N);	00410000
42	1		/* --- R(N,M) = A(M,N) --- */	00420000
43	1	1	DO I=1 TO M;	00430000
44	1	2	DO J=1 TO N;	00440000
45	1	2	R(J,I)=A(I,J);	00450000
46	1	1	END;	00460000
47	1	1	END;	00470000
48	1		RETURN;	00480000
49	1		SCAMATD: ENTRY (R,S,A,M,N);	00490000
50	1		/* --- R(M,N) = S*A(M,N) --- */	00500000
51	1			00510000

GANA 3505 04/14/78 PLIC-VER-033078

PAGE

STMT LEVEL NEST

```

46 1      DO I=1 TO M;
47 1 1      DO J=1 TO N;
48 1 1 2      R(I,J)=S*A(I,J);
49 1 1 4      END;
50 1 1 1      END;
51 1      END MATMLTD;

```

```

00520000
00530000
00540000
00550000
00560000
00570000

```

ATTRIBUTE TABLE

DECL IDENTIFIER

ATTRIBUTES

2	A	(*,*) PARAMETER ALIGNED BIN FLOAT(LONG)
2	B	(*,*) PARAMETER ALIGNED BIN FLOAT(LONG)
3	I	AUTO ALIGNED BIN FIXED(15,0)
3	J	AUTO ALIGNED BIN FIXED(15,0)
3	K	AUTO ALIGNED BIN FIXED(15,0)
3	L	PARAMETER ALIGNED BIN FIXED(15,0)
3	M	PARAMETER ALIGNED BIN FIXED(15,0)
17	MATADD	ENTRY BIN FIXED(15,0)
51	MATG	ENTRY BIN FIXED(15,0)
1	MATMLTD	ENTRY BIN FIXED(15,0)
24	MATSUBD	ENTRY BIN FIXED(15,0)
3	N	PARAMETER ALIGNED BIN FIXED(15,0)
2	R	(*,*) PARAMETER ALIGNED BIN FLOAT(LONG)
2	S	PARAMETER ALIGNED BIN FLOAT(LONG)
45	SCAMATD	ENTRY DEC FLOAT(SHORT)
38	TRANSPD	ENTRY DEC FLOAT(SHORT)

STORAGE REQUIREMENTS.

THE STORAGE AREA (IN STATIC) FOR THE PROCEDURE LABELLED MATMLTD IS 300 BYTES LONG.
 THE PROGRAM CSECT IS NAMED MATMLTD AND IS 2194 BYTES LONG.
 THE STATIC CSECT IS NAMED MATMLTDA AND IS 420 BYTES LONG.

STATISTICS SOURCE RECORDS = 57, PROG TEXT STMTS = 51, OBJECT BYTES = 2194

COMPILER DIAGNOSTICS.

WARNINGS.

1EM05261 1 OPTION MAIN HAS NOT BEEN SPECIFIED FOR THE EXTERNAL PROCEDURE, STATEMENT NUMBER 1.

END OF DIAGNOSTICS.

ELAPSED TIME .05 MINS

WANA 3504 04/14/78 PLIC-VER-033078
VERSION 5.5

OS/360 PL/I COMPILER (F) ON VMOS

PAGE
DATE 78.1

STMT LEVEL NEST

1			MATVECD: PROCEDURE (R,A,B,M,N);	00010000
			/* --- R(M) = A(M,N) * B(N) --- */	00020000
2	1		DCL (P(*),A(*,*),B(*),S) BIN FLOAT(S?);	00030000
3	1		DCL (I,J,M,N) BIN FIXED;	00040000
4	1		DO I=1 TO M;	00050000
5	1	1	R(I)=0.;	00060000
6	1	1	END;	00070000
7	1	1	DO J=1 TO N;	00080000
8	1	1	DO J=1 TO N;	00090000
9	1	2	R(I) = R(I) + A(I,J)*B(J);	00100000
10	1	2	END;	00110000
11	1	1	END;	00120000
12	1	1	RETURN;	00130000
13	1		VECVCD: ENTRY (S,B,M,N);	00140000
			/* --- S = B(N) * R(N) --- */	00150000
14	1		S=0.;	00160000
15	1		DO I=1 TO N;	00170000
16	1	1	S=S+B(I)*R(I);	00180000
17	1	1	END;	00190000
18	1	1	RETURN;	00200000
19	1		VECMATD: ENTRY (R,B,A,M,N);	00210000
			/* --- R(N) = B(M) * A(M,N) --- */	00220000
20	1		DO J=1 TO N;	00230000
21	1	1	R(J) = 0.;	00240000
22	1	1	DO I=1 TO M;	00250000
23	1	2	R(J) = R(J) + B(I)*A(I,J);	00260000
24	1	2	END;	00270000
25	1	1	END;	00280000
26	1	1	RETURN;	00290000
27	1		VECLEND: ENTRY (S,B,N);	00300000
			/* --- S = B --- */	00310000
28	1		S = 0.;	00320000
29	1		DO I=1 TO N;	00330000
30	1	1	S = S + B(I)*B(I);	00340000
31	1	1	END;	00350000
32	1	1	S = SQRT(S);	00360000
33	1	1	RETURN;	00370000
34	1		SCAVECD: ENTRY (R,S,B,M,N);	00380000
			/* --- R(N) = S*B(N) --- */	00390000
35	1		DO I=1 TO N;	00400000
36	1	1	R(I)=S*B(I);	00410000
37	1	1	END;	00420000
38	1	1	RETURN;	00430000
39	1		END MATVECD;	00440000

ATTRIBUTE TABLE

DCL IDENTIFIER

ATTRIBUTES

2 A

(*,*) PARAMETER ALIGNED BIN FLOAT(LONG)

INDEX

Abstraction 13, 21, 35, 48, 50, 200

Adjacency matrix 152, 173, 186, 190, 202, 207, 231, 246, 252

Aggregation 197

Algorithm 2, 11, 13, 15, 16, 17, 18, 52, 169, 184, 194, 195,
200, 202, 204, 208, 209, 213, 214, 215, 216, 219,
222, 224, 225, 226, 229, 230, 233, 234, 244, 249,
250, 252, 254, 264, 267, 270, 272, 275, 276, 277,
278, 280, 281, 282, 284, 288, 294, 298, 321, 323,
324, 336, 350, 351, 352, 353

Almon process 337, 341, 344, 345

Alternatives 137

Analyzer 160

Ancestor 73, 95, 222, 223

Ando 196

Antecedent 201, 203, 214

Application program 1

Arbib 286, 287

Argument 86, 97, 100, 102, 103, 104, 105, 119, 124, 147,
148, 326, 327, 328, 329, 330

Arithmetic expression 83, 86, 116, 117, 118, 124, 125, 126,
127, 129, 192, 277

Artificial Intelligence 22, 27, 215

Assertion 1, 8, 51, 52, 55, 56, 57, 60, 62, 63, 76, 80, 81,
82, 83, 85, 86, 87, 88, 97, 113, 115, 116, 117,
118, 119, 121, 122, 123, 124, 125, 126, 127, 129,
130, 132, 139, 140, 147, 149, 150, 151, 152, 156,

161, 162, 163, 170, 173, 175, 176, 178, 182, 183,
185, 186, 191, 194, 195, 216, 217, 218, 221, 222,
229, 230, 231, 232, 235, 236, 237, 238, 240, 241,
242, 243, 253, 255, 256, 263, 266, 267, 268, 270,
272, 275, 276, 277, 278, 279, 280, 282, 293, 304,
312, 313, 314, 315, 321, 325, 335, 339, 340, 341,
351, 352

Assignment 3, 50, 51, 55, 58, 126, 128, 195

Associative Memory 9, 146, 152, 163, 168, 169, 170, 171,
174, 178, 182, 185, 189, 235, 236, 263, 264, 265,
268, 276, 335

Auto Regressive Schemes 36, 324, 338

Automatic programming 46

Automatic testing 28, 30, 47

Automatic Program Generation (APG) 1, 2, 7, 11, 12, 13, 15,
17, 22, 25, 26, 29, 30, 31, 47, 53, 54, 63, 155,
160, 163, 209, 244, 348, 354

ACRDRA 221

ADJMP 221

ADS 28

ALGOL 38, 163

ANALCMP 229

APL 38, 323

Berner 24

Bersztiss 213

Binary relation 197, 208

Blanks 92

Block angular 351
 Block recursive 11, 18, 41, 55, 200, 252
 Block triangularization 190, 200
 Block-diagonal 198, 200, 207
 Block-triangular 198, 202, 252
 Brookings model 23, 41
 Built in functions 87
 Business 47, 48, 55, 58, 191, 348
 BINARY 107
 BIT 107
 Causality 51, 63, 162, 190, 195, 196, 198, 199, 200, 245
 Chang 30
 Character set 90
 Circuits 212, 213, 214, 244
 Clustering 249
 Code generation 11, 15, 16, 19, 161, 166, 174, 185, 189,
 191, 192, 235, 251, 268, 269, 271, 273, 276, 348
 Coefficients 36, 68, 108, 162, 253, 299, 305, 324, 329, 333,
 339, 340
 Collinearity 337, 341
 Command language 50, 146, 322
 Comments 92
 Common data area 239
 Communication 3, 14, 16, 21, 48, 144, 145, 146, 156, 158,
 175, 178, 193, 296, 323, 348, 350
 Commutativity 280, 282, 283
 Compilation 12, 155, 192, 252

Compiler-compiler 163

Completeness 10, 13, 29, 31, 57, 60, 63, 84, 144, 152, 161,
174, 175, 177, 184, 209, 251, 325, 341

Component 11, 34, 35, 51, 78, 90, 97, 109, 158, 159, 163,
165, 211, 212, 229, 233, 244, 266, 342, 343, 354

Composition 58, 70, 83, 88, 89, 97, 130, 143, 145, 150, 166,
170, 174, 180, 182, 185, 199, 296, 320, 353, 354

Compound 56, 195, 198, 212, 237

Computation Description 60, 62, 89, 95, 118, 130, 185

Condensation 56, 211, 212, 235

Consistency analysis 13, 29, 156, 167, 174, 184, 325

Constant adjustment 36, 143

Constant Value Shares 321

Constants 92, 93

Constituent 198, 208

Contract 286, 287, 292, 297, 302

Convergence 63, 64, 122, 126, 148, 149, 150, 241, 245, 250,
251, 255, 257, 258, 260, 261, 262, 263, 271, 291,
321, 347, 353

Cooperative computation 13, 14, 18, 25, 286, 289, 292, 297

Covariance 32, 324, 338

Cross-links 222, 223, 224

Cross-reference 11, 56, 144, 152, 186, 187, 321

Cross-sectional 26, 354

Cycles 10, 15, 18, 19, 63, 144, 162, 184, 194, 195, 204,
209, 212, 213, 218, 224, 251, 268, 270, 275, 276,
340, 350, 351, 352, 353

Cycles analysis 10, 18, 19, 63, 150, 162, 170, 174, 183,
184, 189, 194, 195, 208, 209, 213, 235, 242, 243,
263, 340, 350, 351, 352, 353

CHARACTER 106

CDPPP 24

CLASFY 250

COBOL 6, 58, 106

COUNT 88

CSMP 39, 42

Data management 6, 13, 20, 25, 146

Data structure 6, 8, 26, 44, 45, 48, 49, 50, 51, 52, 57, 58,
59, 63, 69, 70, 84, 86, 116, 118, 121, 127, 129,
145, 191, 304, 348, 349

Data Area 239, 241, 242, 266, 267, 268, 272

Data Description 6, 8, 14, 29, 52, 57, 58, 60, 69, 70, 73,
74, 77, 86, 89, 97, 122, 127, 152, 185, 191, 304,
305, 311, 313, 314, 349

Data Description Language (DDL) 29, 160

Database 8, 13, 27, 36, 37, 46, 49, 51, 53, 57, 60, 63, 78,
87, 95, 145, 146, 158, 174, 286, 287, 289, 291,
299, 300, 302, 304, 305, 313, 314, 315, 348, 349

Decision support systems 46

Decomposable 197, 198, 245, 246, 352

Decomposition 10, 15, 18, 244, 250, 276, 351, 352

Depth First Search (DFS) 215, 216, 222, 223, 229, 230, 233,
280, 281

Descendant 76, 79, 106, 176, 216, 217, 222

Derivation tree (DT) 240, 277, 278, 280, 285

Determinate 197

Differential equations 42

Directed Graph, Digraph 10, 11, 155, 156, 173, 175, 180,
182, 189, 194, 207, 209, 210, 211, 213, 216, 218,
223, 230, 235

Directory entry 170, 171, 172, 182, 236, 237, 238

Disconnected 201

Distributed computation 13, 287

Distributed Lags 36, 324, 337, 341, 344, 352

Documentation 11, 19, 29, 48, 51, 55, 56, 68, 79, 80, 81,
109, 112, 130, 136, 144, 145, 152, 158, 185, 192,
235, 237, 296, 320, 322

Donovan 46

Dynamic multiplier 141, 142

Dynamic simulation 13, 42, 48, 68, 81, 113, 136, 139, 141

DECIMAL 107

DYNAMO 39, 41, 42

Econometric 22, 23, 24, 25, 36, 41, 43, 48, 57, 141, 244,
249, 250, 253, 254, 255, 288, 296, 322, 323, 335,
336, 352

Economics 4, 7, 12, 17, 21, 22, 34, 36, 41, 43, 47, 55, 64,
196, 256, 288, 296, 299, 304, 305, 306, 353

Editor (See Text Editor)

Edge-digraph 214

Efficiency 15, 27, 63, 190, 192, 213, 214, 249, 250, 252,
256, 261, 276, 338, 348

Eigenvalues 260, 261, 328, 343

Eigenvectors 328, 343

Endogenous variables 34, 35, 66, 72, 81, 130, 139, 140, 141,
142, 151, 161, 162, 196, 197, 199, 205, 218, 254,
255, 262, 271, 275, 293, 297, 299, 302, 311, 312,
314, 354

Entities 39, 40, 69, 70, 73, 83, 103, 104

Equation (See Simultaneous equations)

Equivalence relation 56, 150, 210, 214, 215

Estimation techniques 5, 11, 16, 36, 49, 64, 322, 323, 325,
335, 336, 341

Events 39, 40, 49, 70

Execution 12, 15, 144, 145, 155, 192, 208

Exist (EXIST) 62, 117, 216, 217

Exogenous variables 34, 35, 66, 69, 72, 130, 141, 142, 162,
249, 253, 289, 290, 293, 296, 297, 298, 299, 302,
311, 312, 354

Expert system 28, 29, 287, 353

Extended Data Area 239, 240, 242, 243, 266

EBNF 89, 159, 164, 240

EBNF/WSC 160, 163, 164, 166, 168

EEC 24

EIGVAL 328

EIGVEC 328

EXPON 332

Fault isolation 30

Feedback 15, 37, 41, 145, 156, 195

Field (FIELD) 45, 62, 70, 72, 73, 76, 77, 78, 79, 80, 81,
83, 88, 97, 103, 104, 105, 106, 108, 113, 116,
117, 118, 119, 124, 137, 147, 182

File (FILE) 28, 36, 39, 49, 59, 62, 68, 69, 72, 73, 76, 77,
78, 79, 81, 83, 88, 97, 102, 103, 104, 105, 106,
108, 109, 110, 112, 113, 114, 115, 116, 130, 131,
137, 140, 147, 152, 158, 175, 182, 185, 191, 222,
305, 311, 312, 313, 315

Fisher 196, 353

Fixed point 255

Flow-chart 11, 51, 56, 58, 155, 189, 191, 225, 251, 252

For-each (FOR-EACH) 80, 83

Forecasting 22, 23, 25, 31, 45, 48, 141, 253, 296

Forrester 41, 42

Fragment 211, 213, 215

Fronde 222, 223, 224

Full Information Maximum Likelihood 36, 65, 341

Function (FUNCTION) 16, 18, 19, 26, 35, 37, 48, 49, 84, 86,
87, 88, 108, 117, 122,, 123, 124, 140, 146, 155,
162, 171, 174, 199, 221, 241, 254,, 262, 280, 281,
293, 297, 298, 312, 321, 324, 325, 326, 327, 328,
329, 330, 331, 332, 333, 336, 338, 344, 347, 350,
354

FIXED 107

FLOAT 107

FORTRAN 22, 38, 40, 41, 42, 43, 44, 50, 244, 323

Garfinkel 22

Gauss-Seidel 41, 63, 124,, 125, 151, 195, 253, 254, 255,
 257, 260, 261, 262, 271, 272, 276
 General purpose programming language 38
 Generalized Least Squares (GLS) 324, 337, 338
 Gibb 28
 Graph theory 156, 195, 210, 353
 Group (GROUP) 45, 70, 72, 73, 76, 77, 97, 104, 105, 108,
 147, 237
 GAMMA 332
 GASP 39, 40, 41, 42
 GPSS 39, 40, 41
 Hardware 4, 5, 31, 52, 97
 Header (HEADER) 60, 89, 93, 96, 113, 162, 183 , 185, 241,
 304 , 305, 313, 314, 325, 339, 340
 Help (HELP) 146, 147, 148
 Heuristic 22, 249, 276
 Hildreth-Lu 338
 Hoerl 346
 Host system 5, 6, 12
 Host Language 4, 5, 192
 Identity 1, 35, 66, 249, 290, 291, 304
 Impact multiplier 142
 Inconsistencies 2, 10, 28, 53, 158, 350
 Incrementality 56, 200, 209
 Indecomposable 198, 199, 202, 208
 Independence 14, 55, 56, 57, 59, 86
 Initial (INITIAL) values 63, 122, 125, 126, 241, 271, 320

Inner iteration 148, 149
Input-output 9, 11, 45, 50, 55, 56
Input-output matrix 325, 333
Instrumental Variables 36, 324, 337
Integrability 2, 3, 7, 23, 24, 57
Integration 2, 14, 18, 23, 24, 25, 26, 57, 200, 302, 313,
323, 348, 350, 354
Interactiveness 28, 31, 37, 43, 44, 48, 50, 144, 145, 150,
156, 174, 175, 178, 193
Interface 6, 7, 14, 15, 16, 26, 44, 141, 158, 174, 293, 335,
348, 353, 354
Interim (INTERIM) 62, 80, 81, 84, 118, 119, 120, 130, 133,
135, 137, 140, 143, 147, 175, 182, 185, 186, 192,
229, 305, 311, 314, 333, 334, 340
International outlook 23
Inverse 280, 281, 283, 294, 327, 329, 333, 338, 342, 344
Iteration 83, 86, 129, 151, 184, 189, 190, 194, 200, 244,
251, 252, 253, 256, 257, 261, 262, 268, 272, 291
Iteration Analysis 31, 48, 189, 190, 191, 209, 251
Iterative methods 10, 18, 22, 37, 41, 56, 63, 64, 125, 150,
195, 208, 241, 254, 255, 261, 267, 271, 275, 337,
338, 341, 346, 347
Iterative Least Squares (ILS) 341
IRS 30
ISDOS 28
ITER 189
Jacobi method 255, 256, 260, 261

Johnson 213
 Johnston 336
 Jungle 223, 233
 Kennard 346
 Klein 41, 64, 67, 78, 81, 129, 136, 142, 186, 336, 349
 Klein-Golderberg model 41
 Lags 13, 36, 65, 84, 130, 139, 140, 141, 151, 162, 253, 254,
 291, 302, 314, 324, 337, 341, 344, 352
 Length 62, 77, 116, 117, 127, 216, 217
 Lexical analysis 160, 166, 167
 Leontief 333
 Library 11, 19, 56, 87, 95, 124, 130, 304, 320, 332, 349
 Limited Information Maximum Likelihood 36, 341, 342
 Lindley 347
 Linear search 15
 Linking simultaneous assertions 163, 183, 189, 235, 242,
 263, 267
 Loops (See also Cycles) 10, 86, 150, 192, 195, 204, 230,
 249, 275
 Lubin 42, 44
 LES 321
 LINK world trade model 12, 18, 19, 25, 248, 249, 287, 288,
 289, 292, 293, 295, 296, 297, 299
 LINKAS 242
 LOWLINK 224, 233
 Macroeconomic model 23, 34, 49, 51, 64
 Matrix operations 16

Media (MEDIA) 73, 76, 97, 98, 99, 100, 101, 102, 103, 108,
145, 147, 182, 185, 191

Merging simultaneous assertions 162, 163, 189, 235, 243,
263, 267

Metallanguage 159, 160, 163

Metzler 24

Minimal (or Zero order) 200, 201, 202, 205

Model building 1, 5, 6, 12, 16, 17, 23, 25, 31, 38, 43, 44,
45, 48, 49, 50, 51, 55, 56, 87, 161, 191, 251,
260, 296, 322, 325, 348, 349, 350

Model-I 64, 66, 67, 78, 81, 83, 96, 108, 119, 120, 122, 123,
124, 125, 129, 130, 134, 136, 137, 142, 152, 153,
154, 186, 349

Modeling methodology 2, 32, 37, 69, 86, 145, 287

Module (MODULE) 7, 9, 11, 53, 60, 64, 81, 86, 88, 89, 93,
95, 96, 97, 119, 123, 124, 132, 137, 138, 147,
150, 155, 156, 158, 173, 183, 185, 189, 293, 304,
305, 313, 320, 349

Module name 60, 93, 94, 305, 313

Module Description 8, 57, 89, 137, 138, 139, 189, 190

Multiplier analysis 69, 142, 298

MATMLT 326, 334, 344

MATVEC 329, 334

MAXILINK 297

MAXILONG 296, 297, 298, 299

MINILINK 299, 302, 303, 304, 313, 314, 316, 320, 321

MINV 87, 327, 333, 334

MMVERT 329

MODEL 7, 8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 21, 25,
26, 30, 31, 34, 45, 47, 51, 52, 53, 55, 56, 57,
58, 59, 60, 63, 64, 66, 69, 73, 74, 77, 78, 79,
80, 83, 86, 87, 88, 89, 95, 103, 122, 141, 142,
144, 145, 146, 147, 150, 152, 155, 157, 158, 159,
160, 161, 163, 164, 167, 168, 169, 173, 174, 175,
184, 189, 192, 193, 194, 197, 208, 209, 212, 213,
214, 216, 218, 224, 231, 232, 234, 235, 237, 238,
251, 252, 264, 267, 272, 276, 277, 281, 287, 293,
294, 296, 298, 299, 304, 311, 320, 321, 322, 324,
325, 326, 330, 332, 335, 336, 339, 340, 341, 347,
348, 349, 350, 351, 352, 353, 354, 355

MOMENT 328, 330, 331, 343

MPS 23

Names 91

Naylor 31

Network 13, 62, 75, 76, 78, 79, 87, 97, 156, 287

Newton method 41, 63, 124, 195, 254, 275, 276

Non-linear 37, 41, 49, 63, 184, 193, 195, 251, 253, 261,
276, 324, 336, 337, 338, 339, 351

Non-procedural 1, 2, 3, 7, 13, 14, 15, 20, 21, 25, 45, 51,
52, 55, 58, 155, 169, 184, 194, 195, 264, 276,
322, 325, 326, 336, 347, 348

Normalization 15, 18, 230, 251, 252, 255, 256, 258, 259,
260, 275, 276, 277, 278, 279, 280, 282, 283, 284,
285, 312, 340, 342, 343, 353

Numerical methods 184, 212, 263
Nunamaker 29
NBER 43, 323
NOPAL 30
NORMAL 281, 282, 283, 284, 332
NUMERIC 107
Occurrences 78, 79, 80, 83, 104, 105, 107, 116, 166, 213
Off-diagonal 290, 291, 292, 293, 297
One-period simulation 68, 113, 139, 141
Operators 90
Optimization 7, 12, 15, 16, 27, 31, 189, 192, 252, 253, 320,
348
Ordering 22, 208
Ordinary Least Squares (OLS) 36, 324, 329, 333, 335, 336,
337, 338, 339, 341, 345, 346, 347
Orthogonal 342, 344
Output set 256, 275, 276, 353
OPAL 30
Parallel computation 208, 250, 256
Parameter 16, 26, 34, 36, 43, 63, 68, 77, 78, 84, 88, 113,
118, 121, 125, 143, 145, 149, 150, 151, 170, 174,
182, 185, 241, 252, 254, 262, 264, 266, 268, 270,
272, 274, 298, 302, 311, 312, 321, 322, 325, 326,
330, 336, 338, 339, 340, 354
Parent 73, 76, 77, 79, 84, 103, 104, 105, 109, 119, 121,
182, 216, 217
Partition 10, 15, 148, 197, 208, 214, 244, 247, 248, 250,

352

Path 203, 210, 214, 215

Planning 24, 286, 296

Pointer (POINTER) 62, 68, 78, 79, 80, 86, 116, 185, 192,
195, 216, 217, 221, 222

Policy analysis 22, 141, 142, 143, 354

Polynomial distributed lag 324, 337, 344

Power method 343 Powers of adjacency matrix 214

Precedence matrix 173, 175, 178, 182, 194, 197, 209, 216,
218, 219, 221, 231, 242

Precedence Graph 9

Precedence List 152, 173, 174, 178, 186, 188, 189, 216, 221,
222, 321

Precedence Relationships 9, 173, 174, 176, 186, 194, 200

Predecessor 83, 121, 176, 183, 186, 194, 216

Principal Components 341, 342, 343, 344

Processor 2, 7, 8, 9, 10, 11, 12, 14, 15, 16, 17, 18, 28,
30, 51, 52, 53, 55, 56, 57, 58, 60, 77, 78, 80,
81, 96, 144, 145, 148, 152, 155, 156, 157, 158,
159, 161, 167, 174, 178, 182, 183, 192, 194, 212,
214, 270, 272, 294, 320, 325, 326, 335, 340, 348,
349, 350

Programming Languages 3, 6, 21, 22, 27, 30, 43, 44, 45, 52,
55, 56, 83, 128, 199, 323

Prywes 27, 31, 43

PICTURE 106, 113

PL/1 3, 6, 11, 12, 14, 29, 30, 38, 45, 50, 53, 56, 58, 87,

88, 90, 101, 106, 155, 166, 189, 191, 192, 219,
224, 226, 251, 252, 253, 274, 283, 284, 320, 323,
330

PLANETS 44, 322

PROTOSYSTEM 29

PSL/PSA 29

Qualifier 55, 56, 79, 80, 95, 104, 105, 116, 117, 118, 119,
122, 148, 149, 163, 183, 184, 243, 263, 267, 268,
269, 270

Question-Answering 16

QED (See also Text Editor) 146

Random variates 87, 332

Randomness 56

Reachability 201, 203, 204, 214

Record (RECORD) 45, 70, 73, 76, 77, 79, 83, 88, 97, 103,
104, 105, 106, 108, 109, 136, 147, 175, 183

Refer (REFER) 95, 96, 147, 313, 314, 315, 349

Reference 60, 62, 68, 69, 72, 83, 87, 93, 95, 103, 116, 123,
124, 150, 162, 165, 166, 184, 185, 313

Regression analysis 16, 18, 19, 322, 325, 333, 337, 339, 340

Relaxation methods 262, 264

Residual check 36, 68, 113, 139, 140, 354

Retrieval 9, 32, 145, 146, 152, 158, 168, 169, 170, 171,
237, 238, 242, 249

Ridge regression 341, 344, 346

Rin 155, 213, 251

Rudd 275

RANK 189, 190 68, 69, 75, 76, 80, 81, 93, 94, 102, 113,
 RAPE 44 119, 130, 143, 147, 149, 293, 305
 RAS adjustment 292, 321
 RETR#E 170, 171 223
 RETREVE 169, 171 10, 11, 15, 17, 28, 29, 30, 34, 36, 52, 53,
 RETRNAS 170, 171 62, 66, 76, 77, 78, 83, 88, 89, 95, 118,
 RETRPRX 170, 172 14, 130, 132, 137, 140, 141, 143, 144, 145,
 Self-contained 197, 198, 199, 200 162, 164, 166, 167, 173,
 Semantic 9, 159, 161, 164, 167, 267, 323 185, 186, 216, 251,
 Semi-decomposition 18, 197, 208, 213, 245, 248 15, 316, 320,
 Sensitivity analysis 209, 245, 250
 Sequencing 8, 10, 11, 15, 16, 19, 21, 39, 48, 51, 63, 156,
 Statistical 189, 190, 191, 251, 252, 234, 143, 322, 323, 324, 325
 Sequencing analysis 162, 189, 190, 191, 209, 341
 Set product 214 272, 274
 Sharing 3, 8, 13, 14, 18, 51, 57, 60, 68, 79, 86, 209, 348,
 Steward 2 354 249, 275
 Shastry 14, 277 tions 35, 62, 63, 108
 Shiller method 341, 344, 345, 346, 347 59
 Simon 196 ntly 163, 169, 170, 171, 172, 185, 221, 236, 237,
 Simulation 1, 4, 5, 11, 16, 17, 18, 20, 231, 36, 38, 39, 40,
 Strongly- 41, 42, 45, 47, 48, 55, 63, 64, 67, 68, 2169, 70,
 80, 89, 113, 130, 137, 139, 144, 150, 186, 197,
 253, 299, 300, 302, 305, 311, 313, 314, 320, 322,
 Structural 354 analysis with Substitution (SWS) 275, 276
 Simultaneous equations 10, 15, 37, 55, 56, 63, 64, 122, 125,
 126, 144, 148, 149, 150, 151, 152, 154, 162, 163,

183, 184, 185, 189, 191, 193, 195, 208, 209, 212,
216, 218, 235, 241, 242, 244, 248, 249, 250, 251,
252, 253, 254, 262, 263, 264, 265, 267, 271, 320,
321, 325, 335, 341, 350, 353

Singularity 327, 329

Skeleton 160

Slysz 43

Smith 347

Software 1, 5, 12, 13, 14, 17, 20, 26, 29, 31, 43, 46, 322,
325, 348, 349, 350, 354

Solution (SOLUTION) 4, 5, 7, 10, 11, 14, 15, 16, 18, 20, 25,
26, 28, 41, 51, 63, 64, 68, 69, 112, 113, 122,
124, 126, 130, 131, 136, 139, 141, 142, 150, 151,
199, 200, 212, 230, 245, 248, 252, 253, 254, 255,
256, 257, 258, 266, 268, 270, 271, 274, 289, 291,
293, 296, 299, 300, 302, 304, 305, 311, 312, 314,
315, 325, 339, 340, 347, 350

Solution procedures 4, 5, 7 8, 11, 14, 15, 16, 18, 20, 26,
37, 41, 45, 49, 63, 64, 122, 124, 125, 149, 151,
152, 185, 190, 191, 192, 208, 241, 251, 252, 254,
261, 262, 263, 264, 267, 268, 270, 271, 272, 325,
350, 353, 354

Source (SOURCE) 8, 34, 60, 63, 76, 79, 80, 81, 86, 88, 113,
121, 122, 123, 135, 137, 139, 152, 160, 161, 162,
169, 170, 175, 176, 182, 183, 191, 192, 217, 218,
224, 229, 230, 238, 241, 254, 293, 313, 315, 321,
339, 340

Source file 3, 68, 69, 75, 76, 80, 81, 93, 94, 102, 113,
118, 119, 130, 143, 147, 185, 293, 305

Soylemez 21, 244, 275

Spanning forest 223

Specification 9, 10, 11, 15, 17, 28, 29, 30, 34, 36, 52, 55,
57, 60, 62, 66, 76, 77, 78, 83, 88, 89, 96, 118,
123, 124, 130, 132, 137, 140, 141, 143, 144, 145,
152, 155, 158, 159, 161, 162, 164, 166, 167, 173,
174, 176, 178, 182, 183, 184, 185, 186, 216, 251,
293, 294, 299, 304, 306, 311, 313, 315, 316, 320,
325, 348, 349, 353

Standardization 25

Statistical analysis 16, 18, 19, 32, 43, 322, 323, 324, 325

Statistical inference 322

Step size 262, 272, 274

Steuert 353

Steward 244, 249, 275

Stochastic equations 35, 62, 63, 108

Storage and Retrieval Subsystem 168, 169

Storage entry 163, 169, 170, 171, 172, 185, 221, 236, 237,
238, 241, 242, 243, 263, 267, 268, 270, 272

Strongly-connected-components 64, 201, 210, 211, 212, 213,
214, 215, 216, 218, 222, 223, 224, 229, 230, 234,
235, 242, 244, 270, 276, 352

Structural Analysis With Substitution (SWS) 275, 276

Subscripts (SUBSCRIPT) 13, 62, 66, 80, 82, 83, 84, 85, 86,
118, 121, 137, 147, 162, 173, 176, 183, 184, 185,

186, 189, 190, 192, 209, 217, 221, 222, 293, 305,
311, 314, 333, 334

Successive Over Relaxation (SOR) 253, 254, 262, 272, 274

Successor 173, 186, 194, 216

Symbolic analysis 22, 209, 250

Syntax Analysis Program (SAP) 159, 160, 161, 162, 163, 164,
166, 167, 168

Syntax Analysis Program Generator (SAPG) 159, 160, 164

SIMSCRIPT 39, 40, 41

SIMULATE 39, 41

SODA 29

SOLVEGS 267, 268, 274

STORE 169, 170

STRGCON 222, 225, 230

SUM 87

Taplin 24

Target (TARGET) 8, 34, 45, 60, 63, 80, 81, 86, 121, 123,
125, 126, 135, 137, 161, 162, 175, 182, 183, 191,
217, 229, 230, 241, 252, 254, 262, 271, 275, 276,
278, 279, 280, 281, 282, 293, 313, 315, 339, 340

Target file 8, 68, 80, 81, 93, 94, 95, 109, 112, 113, 118,
119, 130, 147, 183, 185, 293, 305

Tarjan 222, 224, 226

Taylor series expansion 41, 336

Tearing 18, 213, 244, 250, 352

Teichroew 28, 42, 44

Test (TEST) value 126, 151

Text editor 53, 63, 141, 146, 158, 174

Three-stage Least Squares (3SLS) 36, 341

Time-series 26, 44, 49, 57, 68, 69, 70, 71, 72, 77, 79, 111,
113, 137, 305, 311, 329, 354

Tinaztepe 30

Tolerance 1, 57, 60, 161, 329, 349

Topological Sorting 11, 189, 209

Trade matrices 289, 291, 292, 298, 302, 314

Transferability 4,14

Transformation 5, 32, 36, 49, 57, 118, 127, 155, 293, 294,
322, 324, 326, 337, 338, 339, 340, 341, 354

Transition matrix 160, 167, 246, 247

Transitive closure 203

Transmission 25

Two-stage Least Squares (2SLS) 36, 324, 341, 342

TAG 28

TRACE 343

TRANSP 87, 327, 334, 344

TROLL 43, 44, 323, 336, 338

TSO 146

TSP 44, 323

United Nations 24

UNCTAD 24

UNIFORM 332

UPDATE 169, 172

Variance 334, 342, 343, 347

Vector operations 16

AD-A066 192

MOORE SCHOOL OF ELECTRICAL ENGINEERING PHILADELPHIA P--ETC F/6 9/2
USE AND EXTENSION OF AN AUTOMATIC PROGRAM GENERATOR FOR MODEL B--ETC(U)
OCT 78 J L GANA

N00014-76-C-0416

UNCLASSIFIED

78-02

NL

6 OF 6

AD
A066192

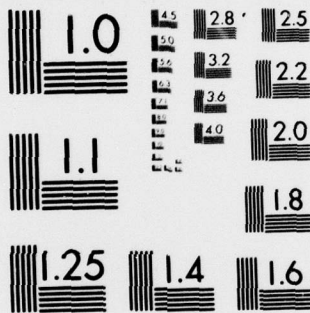


END
DATE
FILMED

5-79

DDC





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Vector space 214

Waifield 214

Warshall 204, 214

Weakly connected 210

Weighted adjacency matrix 173, 194

Welfare function 352

Wharton model 23

World model (See also LINK world trade model) 7, 12, 18,
19, 24, 57, 248, 287, 288, 296, 299

WEFA 24

XPL 160

Bibliography

- [ADS 68] ADS, National Cash Register Co., 1968.
- [AND 63] Ando, A. and Fisher, F., "Near-Decomposability, Partition and Aggregation, and the Relevance of Stability Discussions", Essays on the Structure of Social Science Models, MIT Press, Cambridge, Mass., 1963.
- [AND 76] Ando, A., Norman, A. and Palash, C., "On the Application of Optimal Control to Large Scale Econometric Models", Federal Reserve Bank of New York, Research Paper No. 7517, Aug. 1975, Revised Oct. 1976.
- [ARB 77] Arbib, M.A., "Cooperative Computation and International Planning", Draft Paper, Dept. of Computer and Information Science, University of Massachusetts at Amherst, 1977.
- [ASH 77] Ashcroft, E.A. and Wadge, W.W., "LUCID, A Nonprocedural Language with Iteration", Communications of the ACM, Vol. 20, No. 7, July 1977.
- [BAC 59] Backus, J.W., "The Syntax and Semantics of the Proposed International Language of the Zurich ACM-Gamm Conferences, Proceedings of the International Conference on Information Processing, UNESCO, 1959.
- [BAL 73] Ball, R.J., editor, "The International Linkage of National Economic Models", North-Holland, 1973.
- [BER 71] Berztiss, A.T., Data Structures Theory and Practice, Academic Press, New York, 1971.
- [BER 77] Berner, R.B., "A General Equilibrium Model Of International Discrimination", Ph.D Dissertation, Department of Economics, University of Pennsylvania, 1976.
- [BOS 76] Bosy, M., "A Program for the Design of Procurement

Systems", Tech. Rep. 160. Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge. (Also M.S. Thesis, MIT 1976).

- [BRA 72] Bracy, D.B., "PLANETS, Programming Language for the Analysis of Time Series", (User's Manual), Social Science Computation Center, The Brookings Institution, Washington, D.C., April 1972.
- [BRE 67] Brennan, R.D. and Silberberg, M.Y., "Continuous System Modeling Programs", IBM Systems Journal, Vol. 6, No. 4, Dec. 1967.
- [CAR 73] Cardenas, A.F., "Evaluation and Selection of File Organization -- A Model and System", Communications of the ACM, Sept. 1973, pp 540-548.
- [CHA 77] Chang, Y., "Automatic Test Program Generation", Ph.D Dissertation, Moore School of Electrical Engineering, University of Pennsylvania, 1977.
- [CON 63] Conway, M.E., "Design of Separable Transition Diagram Compilers", Communications of the ACM, July 1963.
- [COU 73] Couger, J.D., "Evolution of Business Systems Analysis and Techniques", Computer Surveys, Vol. 5, No. 3, Sept. 1973.
- [DAH 74] Dahlquist, G. and Bjork, A., Numerical Methods, N.J., Prentice-Hall, 1974.
- [DEO 76] Deo, N. and Mateti, P., "On Algorithms for Enumerating All Circuits of a Graph", SIAM J. Comput., Vol. 5, No. 1, March 1976.
- [DON 76] Donovan, J.J., "Database System Approach to Management Decision Support", ACM Trans. on Database Systems, Vol.1, No. 4, Dec. 1976.
- [EIS 73] Eisner, M. and Pindyck, R., "A Generalized Approach to Estimation as Implemented in the

TROLL/1 System", Annals of Economic and Social Measurement, 2/1, 1973.

- [FAD 65] Fadeeva, V.N., "Computational Methods of Linear Algebra", Dover, 1965.

- [FIS 70] Fisher, F.M., "A Correspondence Principle for Simultaneous Equation Models", Econometrica, Vol. 38, No. 1, Jan. 1970, pp 73-92.

- [FRI 71] Friedman, B., "Econometric Simulation Difficulties: an Illustration", Review of Economics and Statistics, 536, 1971 pp. 381-384.

- [GAN 76a] Gana, J.L., Cycle Analysis and Ordering Using a Generalized Sequencing Algorithm", Working Paper, Automatic Program Generation Project, Moore School of Electrical Engineering, University of Pennsylvania, 1976.

- [GAN 76b] Gana, J.L., "Project LINK: Computer Program Documentation", Economic Research Unit, University of Pennsylvania, 1976.

- [GAN 76c] Gana, J.L., "Some Comments on the Solution of Simultaneous Equation Systems using an APG", Working Paper, Automatic Program Generation Project, Moore School of Electrical Engineering, University of Pennsylvania, 1976.

- [GAN 78] Gana, J.L., Hickman, B.G., Lau, L.J., "Alternative Approaches to Linkage of National Econometric Models", LINK Volume III, in Press, 1978.

- [GAR 77] Garfinkel, D., Roman, C., Kohn, M.C., Achs, M.J., Phifer, J., "Application of Heuristic Search Techniques in the Construction of Complex Metabolic Models", Draft paper, Moore School of Electrical Engineering, University of Pennsylvania, 1977.

- [GIB 75] Gibb, K.R., "Automatic File and Module Design", Internal Memo, Bell Laboratories, Jan. 1975.

- [GOR 62] Gordon, G., "A General Purpose Systems Simulator", IBM Systems Journal, 1, 1962.
- [HAR 66] Hartmanis, J. and Stearns, R.E., Algebraic Structure Theory of Sequential Machines. Engelwood Cliffs, N.J., Prentice-Hall, 1966.
- [HAX 75] Hax, A.C. and Martin W.A., "Automatic Generation of Customized Model Based Information Systems for Operations Management", Proceedings on the Wharton Conference on Research on Computers in Organizations, Philadelphia, Oct. 1975, H.L. Morgan, ed.
- [HER 73] Hersley, Rataj, Teichroew and Berg, PSL Manual, ISDOS Working Paper #68, University of Michigan, Oct. 1973.
- [HEW 71] Hewitt, C., "Planner: A Language for Proving Theorems in Robots", International Joint Conference on Artificial Intelligence, London, 1971.
- [HIL 60] Hildreth, C. and Lu, J.Y., "Demands Relations with Autocorrelated Disturbances", Michigan State University, Agricultural Experimental Station, Technical Bulletin 276, Nov. 1976.
- [HOE 70] Hoerl, A.E. and Kennard, R.W., "Ridge Regression: Biased Estimation for Non-orthogonal Problems", Technometrics, 1970, pp 55-67.
- [HOL 64] Holt, C.C., et al, "Program SIMULATE, a User's and Programmer's Manual", Social Systems Research Institute, University of Wisconsin, May 1964.
- [IBM 71] IBM, "The Time Automated Grid System (TAG): Sales and System Guide": GY20-0358-1, May 1971.
- [IBM 74] "OS/VS2 TSO, Command Language Reference", GC-28-0646. IBM Corporation, Poughkeepsie, New York, 1974.
- [IVE 62] Iverson, K.E., "A Programming Language", New York,

Wiley, 1962.

- [JOH 72] Johnston, J., "Econometric Methods", New York, McGraw-Hill, 1972, 2nd. ed.
- [JOH 75] Johnson, D.B., "Finding All the Elementary Circuits of a Directed Graph", SIAM J. Comput., Vol. 4, 1975 pp 77-84.
- [JOH 76] Johnson, K.N. and van Peetersen, A., "Solving and Simulating the LINK System", The Models of Project LINK, J.L. Waelbroeck, ed., North-Holland, 1976.
- [KIV 63] Kiviat, P.J., "GASP - A General Activity Simulation Program", Project No. 90 17-019(2), Applied Research Laboratory, United States Steel, Monroeville, Pennsylvania, July 1963.
- [KLE 50] Klein, L.R., "Economic Fluctuations in the United States, 1921 - 1941", Cowles Commission Monograph II, New York, Wiley, 1950.
- [KLE 74] Klein, L.R., "A Textbook of Econometrics", Prentice-Hall, N.J., 1974, 2nd. ed.
- [KLE 76a] Klein, L.R., "Five-Year Experience of Linking National Econometric Models and of Forecasting International Trade", Quantitative Studies of International Economic Relations, H.Glejser, editor, North-Holland, 1976.
- [KLE 76b] Klein, L.R. and Tishler, A., "Long Run Model of World Trade", Paper Presented at the Project LINK Meeting, University of Pennsylvania, Philadelphia, Mar. 1976.
- [KNU 68] Knuth, D.E., "The Art of Computer Programming," Vol. 1, "Fundamental Algorithms". Addison-Wesley, Reading, Massachusetts, 1968.
- [KRO 62] Krohn, K.B. and Rhodes, J.L., "Algebraic Theory of Machines", Proc. Symp. on Math. Theory of Automata, Polytechnic Press, Brooklyn, N.Y., 1962.

- [LEA 74] Leavenworth, B.M. and Sammet, J.E., "Overview of Non-Procedural Languages", SIGPLAN - Proceedings of a Symposium on Very High Level Languages, Mar. 1974.
- [LIN 72] Lindley, D.V. and Smith, A.F. "Bayes Estimates for the Linear Model, Journal of the Royal Statistical Society, B series, 1972m pp 1-41.
- [LIT 69] Litofsky, B., "Utility of Automatic Classification Systems for Information Storage and Retrieval", Ph.D Dissertation, Moore School of Electrical Engineering, University of Pennsylvania, 1969.
- [MAD 77] Maddala, G.S., Econometrics, McGraw-Hill, 1977.
- [MAL 75] Malhorta, A., "Design Criteria for a Knowledge Based English Language System for Management: An Experimental Analysis", MAC TR-146, MIT Project MAC, Massachusetts Institute of Technology, Cambridge, 1975. (Also Ph.D Dissertation MIT).
- [MAR 62] Markowitz, H.M., et al, "SIMSCRIPT: A Simulation Programming Language", The RAND Corporation, RM-3310, Nov. 1962.
- [MAR 74] Martin, W.A., "OWL: A System for Building Expert Problem Solving Systems Involving Verbal Reasoning", Notes from Course 6871, Massachusetts Institute of Technology, Cambridge, 1974.
- [MCK 70] Mckeeman, W.A., Horning, J.J. and Wortman, D.B., A Compiler Generator, Prentice-Hall, 1970.
- [MET 77] Metzler, L.L.A., "A Multiple-Region Theory of Income and Trade", Econometrica, October 1950, 18:329-354.
- [MOT 77] Motro, A., "Design and Optimization of the Use of Main Memory in Computer Programs", Ph.D Dissertation Proposal, Moore School of Electrical Engineering, University of Pennsylvania, 1977.
- [NAU 60] Naur, P., editor, "Report on the Algorithm

Language ALGOL 60. Communications of the ACM,
Vol. 3., 1960.

- [NAY 68] Naylor, T.H., et al, Computer Simulation Techniques, J. Wiley & Sons, 1968.
- [NIL 71] Nilson, N.J., "Problem Solving Methods in Artificial Intelligence", McGraw-Hill, New York, 1971.
- [NUN 69] Nunamaker, J.F.Jr., "On the Design and Optimization of Information Processing Systems", Ph.D Dissertation, Case Western Reserve University, June 1969.
- [NUN 71] Nunamaker, J.F.Jr., "A Methodology for the Design and Optimization of Information Processing Systems", AFIPS Proceedings, 1971, SJCC, pp 283-294.
- [NUN 72a] Nunamaker, J.F.Jr., et al, "Processing Systems Optimization Through Automatic Design and Reorganization of Program Modules", CSD TR77, Purdue University, Dec. 1972.
- [NUN 72b] Nunamaker, J.F.Jr., et al, SODA, ISDOS Working Paper #36, University of Michigan, Feb. 1972.
- [NUN 76] Nunamaker, J.F.Jr., et al, "Computed Analysis and Design of Information Systems", Communications of the ACM, Vol. 19, No. 12, 1976.
- [ORT 70] Ortega, J.M., Numerical Analysis: a Second Course, N.Y., Academic Press, 1970.
- [PL1 75] PL/1 Reference Manuals, IBM Publications, 1975.
- [PNU 76] Pnueli, A., "A Simple Specification Language to ADP and its Implementation", Working Paper, Automatic Program Generation Project, Moore School of Electrical Engineering, University of Pennsylvania, 1976.

- [PRY 66] Prywes, N.S., "Man-Computer Problem Solving with Multi-List", Proceedings of IEEE, Vol. 54, No. 12, Dec. 1966, pp. 1788-1801.
- [PRY 74] Prywes, N.S., "Automatic Generation of Software Systems -- A Survey", Data Base, Vol. 6, No. 2, Fall 1974.
- [PRY 75a] Prywes, N.S., "Final Report, Automatic Computer Program Generation for Automatic Testing Systems (ATS)", U.S. Army Armament Command, Frankford Arsenal, Philadelphia, June 1975.
- [PRY 75b] Prywes, N.S. and Gana, J.L., "The Use of Computers for Economic Studies of Development", ITCC Rev., 4, No.2(14), Apr. 1975, pp. 15-29.
- [PRY 77a] Prywes, N.S., "Automatic Generation of Computer Programs", Advances in Computers, Vol. 16, Academic Press, 1977.
- [PRY 77b] Prywes, N.S., "Automatic Generation of Computer Programs", Proc. NCC (in press)., 1977.
- [PRY 77c] Prywes, N.S., "MODEL II - Description and User Manual", Contract No. TIR-7 T-62. Internal Revenue Service, Planning and Research Division, Washington, D.C. 1977.
- [PUG 63] Pugh, A.L., "DYNAMO User's Manual. Cambridge, Mass.: The MIT Press, 1963.
- [RAD 72] Raduchel, W.J., "The Regression Analysis Program for Economists", (Reference Guide), Harvard Institute of Economic Research, Harvard University, Cambridge, Mass., Technical Paper No. 10, May 1972.
- [RAM 73] Ramirez, J.A., "Automatic Generation of Data Conversion Programs Using a Data Definition Language", Ph.D Dissertation, Moore School of Electrical Engineering, University of Pennsylvania, 1973.

- [RHO 73] Rhomberg, R.R., "Towards a General Trade Model", The International Linkage of National Economic Models, R.J.Ball, ed., North-Holland, 1973.
- [RIN 76] Rin, N.A., "Automatic Generation of Business Data Processing Programs from a Non-Procedural Language", Ph.D Dissertation, Moore School of Electrical Engineering, University of Pennsylvania, 1976.
- [RUD 64] Rudd, D.F. and Steward, D.W., "On Information Flow and Process design Calculations", American Institute of Chemical Engineers, Las Vegas, Sep. 1964.
- [RUT 76] Ruth, G., "Protosystem I: An Automatic Programming System Prototype", Laboratory for Computer Sciences, Massachusetts Institute of Technology, Cambridge, 1976.
- [SEV 72] Severance, D.G., "Some generalized Modeling Structures for Use in the Design of File Organizations." Ph.D Dissertation, University of Michigan, 1972.
- [SHA 73] Shapiro, B.A., "A Survey of Problem Solving Language Systems" TR-235, Office of Computing Activities, University of Maryland, Mar. 1973.
- [SHA 78] Shastry, S.K., "Interactive Automatic Program Generator", Forthcoming Ph.D Dissertation, Moore School of Electrical Engineering, University of Pennsylvania, 1978.
- [SHI 73] Shiller, R.J., "A Distributed Lag Estimator Derived from Smoothness Priors", Econometrica, Jul 1973, pp 775-788.
- [SIM 53] Simon, H.A., "Causal Ordering and Identifiability", Studies in Econometric Methods, W.C. Hood and T.C. Koopmans, editors, Ch. 3, Wiley, 1953.
- [SLY 74] Slyszy, W.D., "An Evaluation of Statistical Software in the Social Sciences", Communications of the ACM, Vol. 17, No. 6, June 1974.

- [SOY 71] Soylemez, S., "Computer Generated Fortran Programs to Prepare Material and Energy Balances for Process Units", Ph.D Dissertation, Department of Chemical Engineering, University of Pennsylvania, 1971.

- [STE 62] Steward, D.V., "On an Approach to Techniques for the Analysis of the Structure of Large Systems of Equations", SIAM Review, Vol. 4, No. 4, Oct. 1962.

- [STE 65] Steward, D.V., "Partitioning and Tearing Systems of Equations", J. SIAM Numer. Anal., Ser. B, Vol 2., No. 2, 1965.

- [STE 74] Steuert, J., "Generating Submodels of a Simultaneous Equation Model: An Algorithm Using Fisher's Correspondence Principle", Annals of Economic and Social Measurement, 3/3 1974.

- [TAG 68] TAG, Sales and Systems Guide, IBM, GY20-0358-1, 1958.

- [TAP 67] Taplin, G.B., "Models of World Trade", IMF Staff Papers, November 1967.

- [TAR 72] Tarjan, R., "Depth-First Search and Linear Graph Algorithms", SIAM J. Comput., Vol. 1, No. 2, June 1972.

- [TEI 66] Teichroew, D. and Lubin, J.F., "Computer Simulation - Discussion of the Technique and Comparison of Languages", Communications of the ACM, Vol. 9, No. 10, Oct. 1966.

- [TEI 71] Teichroew, D. and Sayani, H., "Automation of Systems Building", Datamation, Aug. 1971.

- [TEI 72] Teichroew, D., "A Survey of Languages for Stating Requirements for Computer-Based information Systems", Fall Joint Computer Conference, 1972.

- [TEI 74] Teichroew, D., Hersley, E.A., Bastarache, M.J., "An Introduction to PSL/PSA", ISDOS Working Paper #86,

Dept. of Industrial and Operations Engineering,
University of Michigan, March 1974.

- [THA 71] Thall, R.M., "A Manual for PSA/ADS: A Machine Approach to Analysis of ADS", ISDOS Working Paper #35, University of Michigan, Oct. 1971.
- [TIN 77] Tinaztepe, C., "FITS-TOP PART OF NOPAL, A Computer Program to Design Test Specifications in the NOPAL Language", Ph.D Dissertation, Moore School of Electrical Engineering, University of Pennsylvania, 1977.
- [TRO 73] TROLL: An Introduction and Demonstration, Computer Operation Activity, National Bureau of Economic Research, Nov. 1973.
- [TSP 73] "TSP - Time Series Processor with WEFA Data Bank", (User's Manual), Economic Research Unit, University of Pennsylvania, Sept. 1973.
- [VIN 78] Vinod, H.D., "A Survey of Ridge Regression and Related Techniques for Improvements Over Ordinary Least Squares", Review of Economics and Statistics, North-Holland, Vol. LX No. 1, Feb 1978
- [WAE 76] Waelbroeck, J.L., editor, The Models of Project LINK, North-Holland, 1976.
- [WAR 62] Warshall, S., "A Theorem on Boolean Matrices", Journal of the ACM, Vol. 9, 1962, pp. 11-12.
- [WAR 73] Warfield, J.N., "Binary Matrices in System Modeling", IEEE Trans. on Systems, Man, and Cybernetics, Vol. SMC-3, No. 5, Sep. 1973.
- [WAR 74] Warfield, J.N., "Towards Interpretation of Complex Structural Models", IEEE Trans. on Systems, Man, and Cybernetics, Vol. SMC-4, No. 5, Sep. 1974.
- [WEI 71] Weil, R.L. and Kettler, P.C., "Rearranging Matrices to Block-Angular Form for Decomposition (and other) Algorithms", Management Science XVIII,

Sept. 1971.

[WIL 64] Williams, J.W.J., "The Elliott Simulator Package",
The Computer Journal, VI, Jan. 1964, pp.
328-331.

[WIN 72] Winograd, R., "Understanding Natural Language",
Academic Press, Massachusetts Institute of
Technology, 1972.